البرمجة بلغة C + +

مدخل إلي

(البرمجة الشيئية وهياكل البيانات)

دكتور
سمير أبو الفتوح صالح
أستاذ ورئيس قسم المحاسبة
كلية التجارة-جامعة المنصورة

دكتور
محمد سعيد شوقي
استشاري النظم وخبير البرامج
معهد الكلي - جامعة المنصورة

٢٠٠٥

صفحة	
i	مقـــدمة
iii	**فهــرس**

مقدمة الكتاب

لغة C وإمتدادها ++C هي اقوى لغات البرمجة التي ظهرت في عالم الحاسبات ، وتعد لغة البرمجة للقرن القادم ، فهي لغة قوية ذات إمكانيات هائلة . ولقد تطورت في السنوات الأخيرة لتتعامل مع طرق البرمجة الحديثة مثل البرمجة شيئية التوجه (الموجهة بالهدف) .

وهذا قد ساعد اللغة على أن تدخل مجالات جديدة في التطبيق تتراوح بين التطبيقات الأدنى مستوى والتي تتطلب معرفة بهندسة الحاسب لا وربما تساعدك على فهم بناء الحاسبات وحتى التطبيقات عالية المستوى والتي لاتتطلب منك أكثر من عدة أسطر لتتعامل مع قائمة كبيرة من البيانات ، وكذا هياكل البيانات المتقدمة .

ومنذ الاصدار الأخير لميكروسوفت فيجوال سي ++ أصبح التعامل معها أسهل كثيرا ، مما يجعلها متاحة لقطاع عريض من المبرمجين الراغبين في إنشاء برامج تعمل في بيئة ويندوز سواء هواة أو محترفين .

ونحن في فصول هذا الكتاب نحاول أن نقدم للقارئ كل مايحتاجه عن هذه اللغة ، نشأتها ، تطورها ، وطريقة إستخدامها . والكتاب شامل لجميع مترجمات لغة سي المتداولة بداية من ANSI C ، BORLAND C ، ++BORLAND C وحتى ++Visual C .

[illegible] التوفيق ،،،،،

المؤلفان


الفصل الأول
مقدمة إلى لغات
البرمجة

الفصل الأول
لغات البرمجة

تمثل البرمجة خطوة أساسية لتوجيه جهاز الكمبيوتر لتنفيذ المهام التي يريدها المستخدم ومن ثم استخراج النتائج المطلوبة .ويمكن القول بأن البرنامج هو مجموعة من التعليمات يتم تنفيذها خطوة بخطوة لتوجيه الحاسب لأداء مهام محددة للتوصل إلي النتائج المرغوب فيها.

المنافع التي تحققها عملية البرمجة لمستخدمها :

- أنها تساعد المبرمج علي فهم أجهزة الحاسب ،فإذا تعلم الفرد كيف يكتب برنامج بسيط فسوف يمكنه من اكتساب المزيد من المعرفة عن طريقة عمل الحاسب.
- أنها تدعم ثقة المبرمج بنفسه وبقدراته وبالتالي يقوم بابتكار وتخليق مجموعة تعليمات جديدة تساهم في حل المشاكل.
- يكتشف المبرمج ما إذا كان لديه رغبة في العمل في مجال البرمجة وما إذا كان يمتلك القدرة التحليلية التي يجب أن تتوفر في المبرمج.

أهم وظائف ومهام المبرمجين:

- تحويل المشكلات إلي تعليمات وإرشادات للحاسب في شكل برنامج .
- تشغيل التعليمات الخاصة بالبرنامج علي جهاز الكمبيوتر للتأكد من سلامتها .
- عمل التصحيحات اللازمة للبرنامج.
- كتابة تقرير عن البرنامج.
- التنسيق مع المبرمجين الآخرين للتأكد من أن البرامج تتلائم مع الغرض منها.

الخطوات اللازمة لإتمام عمليـة البرمجـة (Programming Process) وإعـداد برنامج:

هناك خمس خطوات رئيسية لإتمام عملية البرمجة:

١) تعريف المشكلة Defining The Problem .
٢) التخطيط للحل Planning The Solution .
٣) تكويد البرنامج Coding The Program.
٤) اختبار البرنامج Testing The Program.
٥) تكويد البرنامج Documenting The Program.
٦)

أولاً: تعريف المشكلة : يقوم المبرمج في هذه المرحلة بالآتي:

أ- التعرف علي الوضع الحالي للمنشأة .

ب- تحديد المدخلات وصورها وأنواع البيانات الموجودة بـهـا ،تحديـد المخرجـات وصورها وتحديد العمليات التي تتم علي كل مدخل للوصول إلي المخرج المطلوب

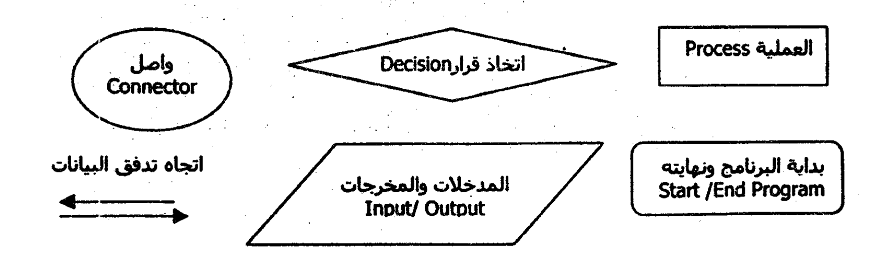
ج- إجراء المقابلات الشخصية مع المديرين بمستوياتهم المختلفـة ومـع العـاملين داخل المنشأة للتعرف علي المشاكل التي تواجه العمل.

د- تحديد احتياجات المستخدم ومتطلباته .

ثانيا: التخطيط للحل: هناك طريقتين عامتين للتخطيط لحل المشكلة :

أ- رسم خريطة تدفق البيانات (Flow Chart) : وهي عبارة عن خطوات متتابعـة لحل المشكلة فهي تمثيل تصويري لحل المشكلة يسهل علي المبرمج معرفة مسار البرنامج المراد تصميمه وهي عبارة عن مجموعة من الأشكال والأسهم.

ومن الأشكال المستخدمة في خريطة تدفق البيانات:



ب- استخدام الرموز الكودية الفرضية (الشفرة الزائفـــة) (Pseudocode): تمثـل برنامج لغوي يعرف ببرنامج الشفرة الزائفة تشبه تعليمات اللغة الإنجليزية وهـي تساعد المبرمج علي تحديد الحلول بدقة ولكنها غير قابلة للتنفيذ علي الحاســب الآلي.

ثالثا : تكويد البرنامج (ترميز البرنامج): تعني ترجمة المنطق الذي تم التوصـل إليـه في خريطة تدفق البيانات أو الرموز الكودية إلي لغة من لغات البرمجة .

رابعا : إختبار البرنامج : حيث يتم وضع البرنـامج موضـع التنفيـذ للتـأكد مـن سـلامته وصحته ولتحديد أي عيوب أو أخطاء في تنفيذه لإعـادة تصحيحـها ، وتتضمـن هـذه المرحلة ثلاثة مراحل:

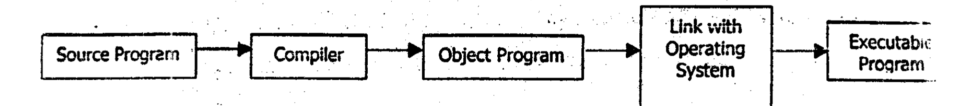
أ- مرحلة فحص القرص (Disk_Checking): في هذه الخطوة يتم عمـل فحـص دقيق للقرص للتأكد من خلوة من الأخطاء وأنه قابل للتشغيل. ويـؤدي هـذا الفحص إلي اكتشاف الخطاء مما يوفر الوقت علي المدى الطويل.

ب- الترجمة (Translating): يتم فيها ترجمة البرنامج المكتـوب بلغـة المصـدر (Source) بواسطة المترجم (Compiler) ويحوله إلي برنامج بلغـة الـهدف (Object Program) لكي يفهمه الحاسب الآلي ويستطيع التعامل معه ثم يتـم ربطه بنظام التشغيل لتحويله إلي ملف ،ويقوم المـترجم بترجمـة البرنـامج بالكامل في وقت واحد .

ويمكن تعريف المترجم بأنه : عبارة عن برنامج يقوم بالآتي:

١. فحص البرنامج اللغوي والنحوي للبرنامج المكتوب بلغة المصدر للتأكد مـن أن لغة البرمجة تم كتابتها بصورة صحيحة .
٢. ترجمة البرنامج المكتوب بلغة المصدر إلي برنامج مكتوب بلغة الآلة ليفهمـه الحاسب بين أجزاؤه الداخلية ويرتبط بكل لغة من لغات الحاسب مترجم خاص به .

وتتم عملية الترجمة كالتالي:



ج- تنقية البرنامج من الأخطاء: إمكانية كشف الأخطاء وتحديد مكانها في البرنامج وتصحيحها وحتى يتم ذلك لابد من اختبار كل جزء في البرنامج في ظل ظروف مختلفة.

خامسا : توثيق البرنامج : وهي عبارة عن وصف مفصل ومكتوب لأصل وطبيعة المشكلة ووصف مفصل للبرنامج والأدوات التي يتم استخدامها مثل خرائط تدفق البيانات والترميز الافتراضي ومواصفات سجلات البيانات والبرنامج .

والمبرمج الحكيم هو الذي يقوم بتوثيق البرنامج خلال قيامه بتصميم البرنامج وتطويره واختباره.

لغات البرمجة (Programming Languages):

تتعدد لغات البرمجة وتتنوع فهناك ما يزيد علي ١٠٠ لغة من لغات البرمجة. وقد تم اختراع لغات البرمجة من أجل أداء وظائف خاصة، وقد استثمرت بعض اللغات لأنها خدمت أغراض خاصة في مجال العلوم والهندسة والتجارة .

ويمكن تعريف لغة البرمجة كالتالي: هي مجموعة من القاعد التي يتم من خلالها إبلاغ الحاسب الآلي بالعمليات التي يجب تنفيذها.

ويمكن تصنيف لغات البرمجة طبقا للغة، فهناك لغات برمجة ذات مستوي أقل وأخري ذات مستوي أعلي، وتختلف كل لغة من اللغات عن الأخرى من حيث درجة سهولتها في الاستخدام والإمكانيات المتاحة بها، وتصنف أجيال لغات البرمجة إلي خمسة أجيال:

١) لغة الآلة (Machine Language) لغة الجيل الأول.

٢) اللغات المركبة أو المجمعة (Assembly Language) لغة الجيل الثاني.

٣) لغات المستوي العالي (High Level Language) لغة الجيل الثالث.
٤) لغات ذات مستوي عالي جدا (Very High Level Language) لغة الجيل الرابع.
٥) اللغات الطبيعية (Natural Language) لغة الجيل الخامس.

أولا : لغة الآلة : تمثل أقل مستوي من لغات البرمجة، حيث ترتكز اللغة علــــي نظـــام الأرقـــام (٠،١) في تمثيل البيانات والتعليمات الخاصة بالبرنامج. ويتم تحويل كل لغـــة مـــن لغـــات البرمجة في النهاية إلي هذه لغة الآلة.

ثانيا : اللغات المركبة أو المجمعة : تعتبر هذه اللغة ذات مستوي أعلي من لغة الآلة، وترتكـز هذه اللغة علي نظام الشفرة الرقمية والاختصارات والرموز (مثلا الرمز C يعبر عن كلمـــة Compare)، ويحتاج المبرمج في ظل استخدام هذه اللغة إلي مترجم لتحويل اللغـــة إلـــي لغة الآلة حتى يستطيع الحاسب فهمها . ويعاب علي هذه اللغة أنها تحتوي علي قدر كبـــير من التفصيل مما يعرض المبرمج لارتكاب الأخطاء نتيجة الملل و التكرار.

ثالثا: لغات عالية المستوي : في ظل استخدام هذه اللغات بدأت أوامر البرمجة تـــأخذ شـــكل مختلف عما كانت عليه قبل ذلك حيث أصبحت شبيهه باللغة الإنجليزية مما جعلتـــها أكـــثر ملاءمة للاستخدام كما أدت إلي توفير وقت وجهد المبرمج كما ساعدته علي إنجاز المـــهام الأكثر تعقيدا . وغالبا ما ترتبط بهذه اللغات مترجم خاص بها لترجمة البرنـــامج المصـــدر إلي برنامج بلغة الآلة التي يتعامل معها الحاسب الآلي بين أجزاؤه الداخلية ومن أمثلة هـــذه اللغات (الكوبول، الفورتران، السي، الباسكال، البيزيك ......الخ).

رابعا : لغات المستوي العالي جدا: وتتسم هذه اللغات بأنها لغات برمجـــة مختزلـــة أي أن العملية التي كانت تتطلب مئات السطور في لغات الجيل الثالث لا تتطلـــب إلا إلـــي ٥-١٠ سطور في لغات الجيل الرابع ولذا يطلق علي هذه اللغات "اللغات الإجرائية: بمعنـــي أنـــها

تقوم بإبلاغ الحاسب بكيفية أداء مهمة ما حيث يتم تنفيذ البرنامج خطوة بخطـــوة" . كمـــا أنها لا تتطلب من المستخدم تزويد الجهاز بالتفاصيل كما كان الحال في لغات الجيل الثالث.

ولذا فإن هذه اللغات تتسم بالإنتاجية حيث قلت عدد الخطوات المســتنفذة فــي تنفيـذ وظيفة معينة أو غرض محدد ،وعلي الرغم من أن لغات الجيل الرابع ســـهلة الاســتخدام بالشكل الذي يجذب العديد من المستخدمين إلا أنها تفتقر إلي عنصري التحكـــم والمرونـــة عن تصور شكل المخرجات المطلوبة.

وتتمثل أهم فوائد لغات الجيل الرابع في الآتي:

١) أنها ذات نتائج يمكن توجيهها والتحكم فيها.
٢) يمكن لها أن تحسن الإنتاجية لأن البرامج سهلة الكتابة.
٣) يمكن استخدامها من قبل المبرمجين وغير المبرمجين مع قليل من التدريب.

وتمثل لغات التساؤل والاستفسار أحد الأشكال المميزة للغات الجيل الرابع والتي تمكـن المستخدم من استبعاد المعلومات من قاعدة البيانات والاستفهام عن أي بيانـــات مطلوبـــة ،ومن أمثلة هذه اللغات (SQL).

خامسا : اللغات الطبيعية : تشبه اللغة الإنجليزية المنطوقة حيث تشبه أوامر العديـــد مـــن العبارات المعروفة في اللغة الطبيعية مما يسهل علي المستخدم استخدامها كما أن اللغـــات الطبيعية تتفوق في سهولة الوصول إلي البيانات ،كما يمكن أن يتعـــرف الحاســـب علـــي المطلوب منه وبالتالي سوف يرسل رسائل تطلب من المستخدم أن يعدل الكلمات إذا كـــان هناك خطأ حتى يستطيع الحاسب أن يفهم المطلوب منه.

ومن التطبيقات الأكثر شيوعا بالنسبة لهذه اللغات التفاعل مع قواعد البيانات .

أسس ومعايير اختيار اللغة المناسبة للبرمجة:

١) بيئة العمل التي يعمل فيها المبرمج حيث قد يقرر مدير المشروع أن يتم اســـتخدام لغـــة معينة من لغات البرمجة.

٢) يجب اختيار اللغة التي تتناسب مع المهمة المطلوبة.
٣) يجب اختيار اللغة التي تتناسب مع طبيعة الجهاز المستخدم .
٤) يجب اختيار اللغة التي تتناسب مع البرامج الأخرى المستخدمة في العمل .
٥) الإمكانيات المتاحة لدي المبرمج من أجهزة ومعدات.
٦) يجب علي المبرمج أن يستخدم اللغة التي يجيدها تماما وله خبرة كبيرة بها.

ومن اللغات التي يستخدمها أغلب المبرمجين لغة البيزيك باعتبار أنها تكون محملة علي أجهزتهم الشخصية.

لغات البرمجة الرئيسية :

١) لغة الفورتران (FORTRAN) :

وهي من لغات المستوي العالي، وهي تعد لغة علمية تم تصميمها لتنفيذ المعادلات المعقدة في مجالات التحليل الاقتصادي والهندسة والرياضيات ومهام البحث العلمي. وهي اختصار لـ (Formula Translator)، وتتميز هذه اللغة بالإيجاز ،ويتكون البرنامج من مجموعة من الأوامر حيث يتم تحديد وتعريف الصور المختلفة من البيانات أثناء التنفيذ من خلال أوامر مثل (Read, Write).

٢) لغة الكوبول (COBOL) :

هي اختصار لـ(Common Business Oriented Language)، أي اللغة العامة للأعمال فقد ظهرت هذه اللغة لتناسب مجال التجارة وقد ظهر من هذه اللغة بعض الإصدارات مثل ANSI_COBOL، COBOL 85 .
وتتميز هذه اللغة بالمرونة حيث يمكن للبرنامج المكتوب بها لنوع معين من أنواع الحاسبات أن يتم تشغيله علي نوع أخر بعد إجراء بعض التعديلات الطفيفة.
وقد تم تصميمها لتتعامل مع احتياجات الأعمال مثل تشغيل الملفات الضخمة وأداء العمليات الحسابية في قطاعات الأعمال مثل الرواتب وحساب الفائدة.

ومن عيوب هذه اللغة أنها بطيئة كما أنها تصيب مبرمجها بالملل وذلك لأنـــها تأخذ وقت طويل في الكتابة.

وينقسم البرنامج في هذه اللغة إلي عدة أقسام وهي:

أ- قسم التعريف : ويتضمن تعريف اســـم البرنــامج ومجموعـــة مـــن التعليقات

ب- قسم البيئة : يتضمن توصيف الحاسب الآلي الذي سينفذ البرنــامج من خلاله وتخصيص كل ملف من الملفات إلي إحدى وحدات الحاسب عند قراءة وطباعة البيانات.

ج- قسم البيانات : يتضمن تفاصيل البيانات التي ســـيتم تشـغيلها فـــي البرنامج.

د- قسم الإجراءات: يتضمن تعليمات محددة للحاسب لتنفيــذ البرنـامج علي نحو منطقي .

٣) لغة البيزيك (BASIC):

تعد من اللغات الأكثر استخداما وتتسم بسهولة التعلـم ،وقـد صممـت لاستخدام الطلاب في بيئة تعليمية وتستخدم في نظم الحاســبات الشـخصية وهي اختصار لــ(Beginners All purpose Symbolic Instruction Code) أي لغة متعددة الأغراض للمبتدئين.

ومن عيوب هذه اللغة أنها تتضمن الكثـــير مـــن القيـــود ولا تصلـــح للاستخدام في المهام المعقدة ،ولكنها تتميز بإمكانية اســتخدامها بواســطة أشخاص غير المبرمجين . وقد ظهر من هذه اللغة بعض الإصدارات مثـــل Microsoft Quick BASIC، Visual BASIC.

٤) لغة الباسكال (PASCAL):

هي من اللغات التعليمية التي تتسم بالنشـــاط والبســـاطة، حيـث تـم استخدامها في أجهزة الحاسب الشخصي كبديل للغة البيزيك. وقد ظهر من هذه اللغة بعض الإصدارات مثل Turbo Pascal التي استخدمت علي نطـاق واسع في مجتمعات الأعمال.

٥) لغة Ada:

يتمثل الهدف الرئيسي لهذه اللغة في تصميم لغة نمطية وتم استخدامها فـي مجال الأعمال التجارية، وقد تم تقديمها برعاية وزارة الدفاع الأمريكـي وIBM و Intel.

ويري البعض أن هذه اللغة معقدة ويري البعض أنها سهلة وتـؤدي إلـي زيادة الإنتاج وأنها لغة تجارية ممتازة. ولكنها تنتشـر فـي مجـال الخدمـات العسكرية لكبر حجمها ودرجة التعقيد الكبيرة بها.

٦) لغة السي C :

هي لغة متعددة الاستعمالات وتتميز عن اللغات الأخرى في صغـر برمجـها بالمقارنة باللغات الأخرى في عدد خطوات البرنامج ،كما يمكن تشغيلها علي أكـثر من نوع من الحاسبات الآلية ولكنها ليست سهلة التعلم حيث تـم ابتكارهـا فـي الأساس ليتعامل معها المبرمجين الموهوبين.

تمكن المبرمج من حل المشاكل البسيطة ولكن لحل المشاكل المعقدة لابـد أن يلم المبرمج باللغة إلماما كاملا.

يمكن تشغيلها علي الحاسبات الشخصية ،وقد ظهر من هـذه اللغـة بعـض الإصدارات مثل ++C.

٧) لغة JAVA:

هي لغة لبرمجة شبكات الاتصالات، وهي تتميز بالسهولة، ويمكن من خـــلال استخدامها تشغيل أجزاء من البرامج في العديد من الأجهزة المختلفة.

لغات البرمجة الموجهة حسب هدف معين (Object Oriented Programming):

يتم فيها برمجة الأشياء (Objects) وخصائصها، ويتمثل Object في وحدة لها تكوين ذاتي تحتوي علي بيانات وحقائق وظائف خاصة بها، ويطلق علي الحقـــائق "الخصـــائص Attributes".

Object: هو فئة أو نوعية آلية يتوافر فيه كافة مواصفات الفئة التي يمثلها.

ومن أمثلة اللغات التي تستخدم في برمجة الأشياء ++C، كما يمكن اســـتخدام لغة تسمي Smalltalk لتدعيم النظام البصري للترجمة حيث يتم إدخال النـــص مـــن لوحة المفاتيح ويتم إجراء كافة التعاملات بواسطة الفأرة. وفي مثل هذا النوع مـــن اللغات لا يحتاج المبرمج أن يكرر التعليمات لكافة المواصفات ممـــا يوفـــر الوقـــت والجهد والمال.

وأخيرا يجب الاشارة الى أن هناك جيل هام مـــن أجيـــال البرمجـــة المتقدمـــة والحديثة وهو جيل الذكاء الاصطناعى حيث ارتكز هذا الجيل علـــى بعـــض اللغـــات المتقدمة مثل لغة الـ Lisp ولغة Prolog .... الخ .


الفصل الثانى
مقدمة إلى لغة
C++

الفصل الثانى
مقدمة إلى لغة ++C

مة :

فـي عـام ١٩٦٠ صممت لغـة Combined Programming CPL Language في جامعة لندن وقد استخدمت فيها بعض تعليمات لغة ALGOL-60 ، وفى عام ١٩٦٧ انبثقت لغة BCPL من لغة CPL على يد مارتن ريتشاردز ، ثم قام تومبسون بتطوير BCPL وسماها لغة B وكل هذه اللغات تعتبر من لغات المستوى الأدنى (Low Level Languages) أى القريب من لغة الآلة مثل لغة التجميع (Assembly).

وفى عام ١٩٧٢ وفى معامل شركة AT&T الأمريكية قام ريتشى بإستباط لغة جديدة من لغة B أخذ منها أحسن تعليماتها وأضاف إليها أوامراً جديدة وأنواعاً جديدة للبيانات وكثيرا من الدوال التى تفيد المبرمج وسميت هذه اللغة بلغة C. ومنذ ذلك التاريخ أخذت لغة C شهرة واسعة لأنها أصبحت تنتمى للغات المستوى الأعلى (High Level Languages) مثل لغة البيسيك والباسكال من حيث سهولة الإستخدام من ناحية ومن ناحية اخرى تنتمى الى لغات المستوى الأدنى من حيث قدرة اللغة على مخاطبة مكونات الجهاز (Hardware)

ومع نجاح وانتشار لغة C انتشرت لهجات كثيرة للمتحدثين بلغة C وكاد يحدث معها ما حدث للغة بيسيك من وجود بيسيك خاص لكل نوع من أنواع الأجهزة سواء كانت أجهزة أى بى ام أو أبل أوكومودور أو سنكلير لولا أن قام معهد القياسات الأمريكية بعملية توحيد لهذه اللهجات المختلفة فأصدر فى عام ١٩٨٣ اللغة القياسية ANSI C ، والجدير بالذكر أن معظم مترجمات لغة C تتوافق مع ANSI C مثل Borland C, Microsoft C, Lattice C.

مميزات لغة C

نورد فيما يلى أهم ما يميز لغة C عن غيرها من لغات البرمجة وهى المميزات التى تدعوك لتفضيلها على غيرها من لغات البرمجة المعروفة وترغبك فى تعلمها واستخدامها.

تتميز لغة C بمجموعة من المزايا مثل:

لغة عامة

أى تصلح لعمل برامج قواعد البيانات والرسومات والحسابات ونظم التشغيل وغيرها

لغة تركيبية (Structured Language)

البرنامج المكتوب بلغة C عبارة عن دالة رئيسية تنادى مجموعة من الدوال الأخرى وكل دالة مجموعة من الأوامر.

تتعامل على مستوى "البت" (Bit Manipulation)

حيث تستطيع أن تقرأ وتكتب وتغير وتقوم بعمليات على مستوى الـ Bit وكما هو معروف فإن الـ Bit هى أصغر وحدة لقياس المعلومات داخل الكمبيوتر وهى جزء من ثمانية أجزاء تعادل فى مجموعها حرف واحد (Byte).

وهذه الميزة جعلتها متخصصة فى بعض مجالات التحكم الآلى والروبوت وبرامج الـ Utility وبرامج معالجة الصور وضغط الملفات واكتشاف الأعطال.

لغة متنقلة (Portable)

أى يمكن للبرنامج المكتوب بلغة C أن يعمل مع اكثر من جهاز مثل IBM أو Apple الأجهزة المتوسطة والكبيرة مع بعض التعديلات الطفيفة.

لغة سريعة

لأن أدوات اللغة تتعامل مباشرة مع الآلة مما يختصر وقت التنفيذ.

لماذا لغة ++C؟

تعتبر لغة ++C لغة بسيطة وسهلة، وذات كفاءة عالية. ومن ثم يمكن استخدامها بمجرد تعلمها. وعلى عكس ما هو شائع، فإن أغلب المبرمجين لايكتبون برامج جديدة، بل كل ما يفعلونه هو أنهم يقضون معظم أوقاتهم فى تحديث وتطوير نسخ البرامج القديمة

تلبية لاحتياجات المستخدم. وتعد البرمجة بأسلوب OOP مثالية فى ذلك. وثمة موضوع رئيسى آخر فى عالم البرمجة بلغة ++C وهو : إعادة استخدام الكود (code reuse). وتعنى هذه العملية ببساطة شديدة : الإستفادة من الكود القديم فى تنفيذ برامج جديدة.

وتنفرد البرمجة بأسلوب OOP ببعض المزايا التى من شأنها تيسير إجراء هذه العملية. وقد أدى ذلك إلى أن أصبحت لغة ++C وسيلة للبرمجة السريعة زهيدة التكلفة. ويتم فى هذا الأسلوب البرمجى إضافة data و behaviors (البيانات والسلوكيات) فى عدة فئات.

فيتم نسخ behaviors فى عملية تعرف باسم inheritance (الإشتقاق). وهى عملية تخضع behaviors بعدها إلى بعض التغييرات فى عملية أخرى تعرف باسم polymorphism (تعدد الهيئات فى برمجة تحديد الأشكال). وسيتم العرض لذلك بمزيد من الشرح والتفصيل ابتداءاً من الفصل التاسع عشر.

وبالإضافة لما سبق، يمكن استخدام لغة ++C فى كتابة أى برنامج وباستخدام أى نوع من الأجهزة طالما أن المستخدم لديه compiler خاص بهذا النوع من الأجهزة. وتخرج البرامج المكتوبة بلغة ++C فى النهاية، والتى تمت كتابتها بشكل جيد، ككيان واحد متكامل يزيد من إمكانيات الجهاز الذى تعمل فيه. ويستطيع المبرمج التحكم فى العمليات ذات المستوى المنخفض فى الجهاز مما يزيد من سرعة ومرونة أداء هذه العمليات. ولا يسرى ذلك على لغة Java التى تقوم فى أساسها على جزء من برنامج يعرف باسم interpreter (المترجم) يعمل على تحقيق التوافق بينها وبين مجموعة من الأجهزة. وقد أدى ذلك إلى افتقاد لغة Java لمستوى السرعة والمرونة المطلوبين للجهاز.

وهناك مجموعة أخرى من اللغات تستخدم فى أساسها لكتابة تطبيقات Windows . وخير مثال على ذلك لغة Visual Basic . وبالرغم من جودة التطبيقات المكتوبة بهذه اللغات، الا أنه يمكن كتابتها بصورة أفضل وأكثر كفاءة باستخدام ++C.

تكوين البرنامج فى لغة C

من الناحية النظرية من الممكن أن تنشئ برنامجاً في لغة C يتكون فقط من الجمل Statements التي تقسوم بكتابتها لكن – عملياً – نادراً ما يحدث ذلك لأن لغة C لا تحمل في داخلها "كتعريف دقيق للغة" أي أوامر لعمليات الإدخال والإخراج I/O .... بمعنى أنه ليس فيها كلمات مفتاحيه تتولى هذه العمليات كما يحدث في لغات أخرى.

لكن مع ذلك فإن لغة C تعوض هذه النقطة من خلال مجموعة من المكتبات Libraries الغنية والتي تحتوي على دوال تستطيع استدعاءها لتنفيذ كل ما تحتاجه سواء في مجال إدخال البيانات وإخراجها أو في مجـــالات أخرى عديدة.

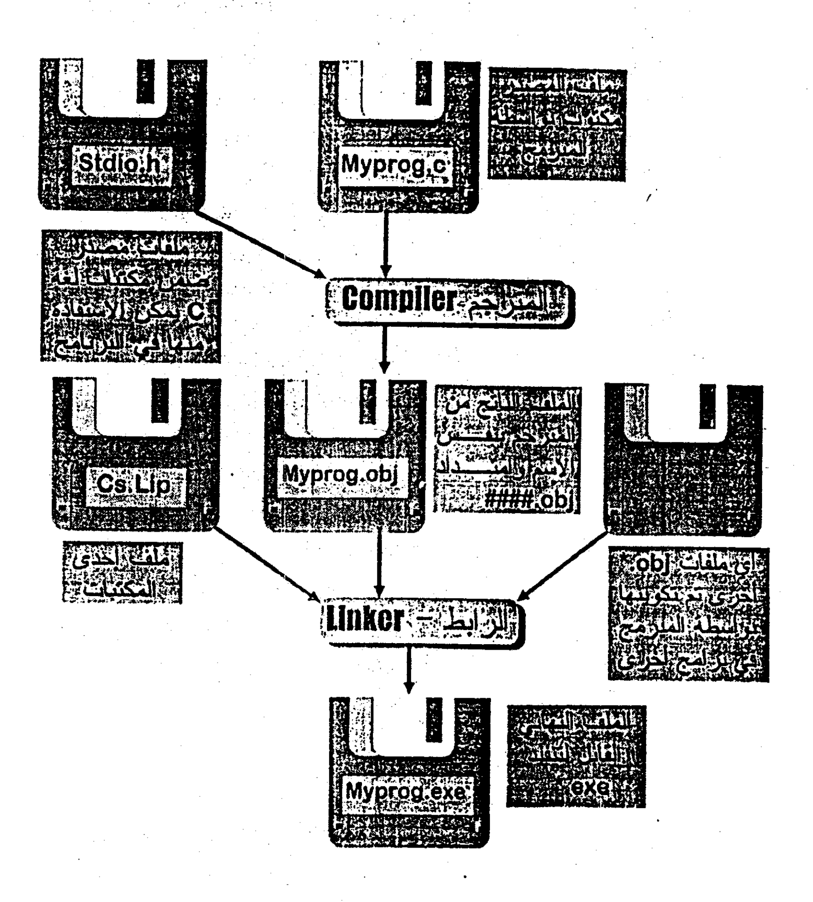
وتتم هذه العملية من خلال ربط البرنامج مع المكتبات فيما يعرف بعملية الربط Linking ويمكن تلخيص خطوات بناء البرنامج في لغة C في الخطوات التالية:

1- كتابة ملف المصدر **Source File** وهو ملف النص الذي يحوي كود البرنـــامج المكتـــوب بواســـطة المبرمج، وهو بمثابة مدخلات عملية الترجمة Compiling ويكون غالباً محتوياً على استدعاء لدوال من مكتبات لغة C .

2- الترجمة **Compiling** وهي عملية ترجمة الكود السابق لكي يتحول إلى ما يفهمه الحاسب بلغة الآلـــة Machine Language والتي يستطيع الحاسب قراءتها وتنفيذها مباشرة ، بالإضافة إلى وضع العلامات الخاصة التي سيتبعها الرابط Linker بعد ذلك لتكوين ملف قابل للتنفيذ (EXE) وتمثل مخرجـــات هذه العملية ملف آخر يسمى (الهدف - Object Code) .

3- الربط **Linking** وهي العملية الثالثة والتي يتم بها ربط البرنامج الهدف مع المكتبات التي يحتاج إليها من بين مجموعة مكتبات C وناتج هذه العملية هو ملف قابل للتنفيذ EXE.*

4 – تصحيح الأخطاء **Debugging** وتتم من خلال تجربة البرنامج بواسطة بيانات تجريبية بحيث تتـــلافى وجود أخطاء منطقية في تكوين البرنامج لأن هذه النوعية لا يكتشفها المترجم أثناء عملية الترجمة.


Stdio.h
Myprog.c
ملفات مصدر ضمن مكتبات لغة C يمكن الاستفادة منها في البرنامج
Compiler المترجم
Cs.Lip
ملف إحدى المكتبات
Myprog.obj
الملف الناتج من المترجم بنفس الاسم والامتداد ####.obj
أي ملفات obj أخرى تم تكوينها بواسطة المبرمج في برامج أخرى
Linker – الرابط
Myprog.exe
الملف النهائي القابل للتنفيذ exe

خطوات تصميم برنامج فى لغة ++C

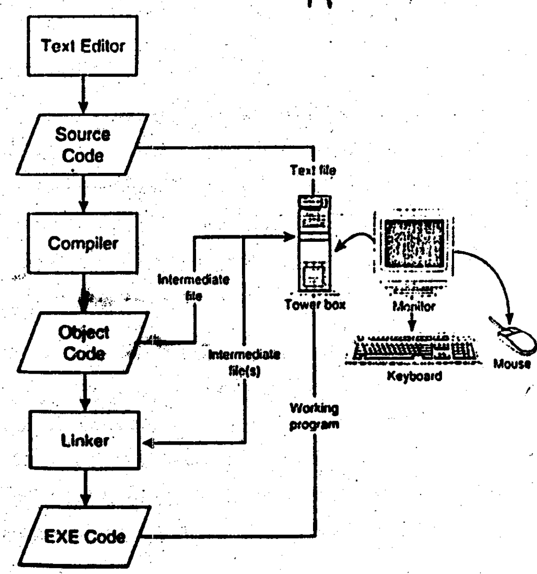
تأخذ عملية تصميم البرنامج فى لغة ++C ثلاث مراحل، ولكن يبقى الهدف الأخير منها جميعًا وضع مجموعة من ال... ات التى يمكن أن يتعامل معها المعالج الحسابى المنطقى(microprocessor) ويفهمها. ففى أجهزة PC -مثلاً- يتم تصميم exe code، التى هى اختصار executable code، وهو الكود الذى يفهمه PC ويتعامل معه. وسنعرض فى الشكل التالى لأهم الخطوات فى compilation process.

وعن المادة فى هذا الكتاب، فقد بنيت على منتجين اثنين فى بيئة عمل لغة ++C، وهما: Borland وMicrosoft، وكلاهما يتبع نسقاً متشابهًا تمام التشابه. ويشتمل كل منتج منهما على text editor (محرر نصوص)، وcompiler، وlinker (الرابط) وعلى الرغم من أن الحديث ينصب هنا على Borland وMicrosoft فقط، فهما ليسا سوى أكبر الشركات المنتجة ويوجد غيرهما الكثير. وأيًا كان نوع المنتج المستخدم فى البرنامج، وأيًا كانت الشركة المنتجة له، فلابد وأن تتوافر فيه العناصر الثلاثة السابق ذكرها، والتى لكل منها وظيفة تؤديها.

(١) text editor: يُستخدم فى أساسه لكتابة البرامج المصممة بلغة ++C مثله فى ذلك مثل معالج الكلمات (word processor). ويُعرف النص الناتج باسم source code.

(٢) compiler: وهو عبارة عن جزء من برنامج، وظيفته البحث عن الأخطاء التى قد توجد فى source code، ثم تحويلها إلى جزء من كود آخر يعرف باسم object code.

(٣) linker: وهو عبارة عن جزء آخر من البرنامج، وظيفته أخذ object code واستخدامه فى ربط مجموعة روتينات جاهزة وذات مستوى منخفض؛ وذلك لإنتاج exe code الذى سوف يعمل فى الكمبيوتر. ويقوم linker - فضلا عن ذلك- بتحديد البيئة التى يعمل فيها هذا البرنامج المصمم بلغة ++C سواء كانت بيئة DOS أو Windows، أو أى بيئة تشغيل أخرى مثل UNIX.



مسار تنفيذ compilartion Process فى لغة ++C

واليك المزيد من التفاصيل الخاصة بهذه العملية:

(أ) text editor

ويُقصد به كل ما يشاهده المستخدم على واجهة compiler لغة ++C . وهو عبارة عن معالج كلمات يمكن من خلاله إدخال نص للبرنامج، ليكون بذلك قد تم تصميم نص للبرنامج بلغة ++C. ويُعرف النص الناتج باسم source code. ويُستخدم أيضًا في معالجة النص الناتج وحفظه وتحميله تمامًا مثل أى معالج كلمات آخر. (ويُختصر text editor عند مستخدمى لغة ++C إلى كلمة editor فقط).

(ب) Compiler

وهو عبارة عن مدقق املائى (spell checker) يفهم كود اللغة المكتوب بها البرنامج ويتعامل معها. فمثلاً فى حالة البرنامج المكتوب بلغة ++C، يقوم المستخدم بالضغط على خيار compiler الخاص بهذه اللغة للتعامل معها. وعند وجود أية أخطاء، سوف يصدر compiler قائمة بها.

وعادة ما يطلق على أخطاء هذه المرحلة اسم compilation errors. وتأتى هذه الأخطاء نتيجة خطأ ما فى كتابة الكود بلغة ++C. ولذلك، فهى تُعرف أيضاً من الناحيه الفنية باسم syntax errors (أخطاء التركيب والصياغة).

(ج) Object Code

وهو عبارة عن جزء جاهز من الكود يقوم compiler بتصنيفه وتجميعه فى حالة ما لم يكن هناك خطأ فى تركيب الجملة البرمجية. ويستخدم هذا الكود المجمع فى المرحلة التالية لذلك فى compilation process. ويُعتبر object code ترجمة لجزء source code، ويتم فيه وضع معلومات عن الوضع النهائى فى العملية التالية. (ويختصر مستخدمو لغة C++ object code إلى obj أو objy code).

(د) Linker:

فى حالة عدم حدوث أية أخطاء فى مرحلة التجميع والتصنيف، فمن الممكن تنفيذ برنامج عاجل من خلال تحويل object code (أو ربما عدد من الكود معاً) إلى برنامج عامل آخر. ويقوم Linker بتصفح واستعراض object code وتجميع برنامج عامل من libraries (مكتبة الملفات) التى تشحن مجاناً مع حزمة برامج C++. ويتم الإشارة إلى أى خطأ من الأخطاء التى قد تقع فى هذه المرحلة باسم Linker errors. وهى أخطاء من الصعب جداً اكتشافها، ومن ثم لا بد من أن يكون المستخدم حذراً ومستعداً لها.

(هـ) Exe code:

وهى الخطوة الأخيرة والفعالة التى تتمثل فى تنفيذ البرنامج وبيعه. وذلك بعد التأكد خلوه من الأخطاء. ويتم بيع exe code، أما source code وobject code فهما خاص بالمبرمج.

الدروس المستفادة:

- تعتبر C++ لغة حديثة فى عالم البرمجة وتتميز بأنها رفيعة المستوى وذات كفاءة عالية فى الأداء. ولأنها صورة متطورة من لغة C، فهى تنتمى لمجموعة أرقى منها وأكثر تطوراً.
- يخرج البرنامج المصمم بلغة C++، والمكتوب بشكل جيد، ككيان كامل متكامل يزيد من إمكانيات الجهاز الذى يعمل فيه. ولذلك، فعادة ما يكون البرنامج المصمم بلغة C++ ذا كفاءة عالية فى الأداء تفوق كفاءة البرامج المكتوبة بلغات أخرى مغايرة كلغة Java وVisual Basic.
- يتشابه text editor. فى وظيفته مع معالج الكلمات؛ فهو المسؤول عن كتابة نص البرنامج بلغة C++ ليتم إرساله بعد ذلك إلى compiler.
- يعتبر source code الكود الرئيسى فى البرنامج المصمم بلغة C++. ولابد من ترجمته إلى object code ليتم الربط بينهما لإنتاج exe code أو exe، وهى النسخة الأخيرة من البرنامج التى سيتم طرحها للبيع فيما بعد.

• يعتبر compiler برنامجًا يشبه فى وظيفته المدقق الإملائى فى معالج الكلمات.
ويقوم compiler بتفقد أى خطأ قد يوجد فى source code، ثم يحول هإلى كود آخر يعرف باسم object code.

• يعتبر Linker أيضاً برنامجًاوظيفته أخذ object code من compiler ليستخدمه فى الربط بين مجموعة من الروتينات الجاهزة ذات المستوى المنخفض. وذلك من أجل إنتاج exe code الذى يمثل الجزء الفعال فى البرنامج الذى سيتم بيعه فيما بعد.

التعامل مع البرنامج

فيما يلى نوضح كيفية التعامل مع البرنامج :

- **لحفظ ملف البرنامج اضغط F2 أو إختر Save من قائمة File**
- **لترجمة وتنفيذ البرنامج اضغط على المفتاحين Ctrl+F9 أو ALT+R ثـم على الاختيار الاول من القائمة نضغط مفتاح الادخال**
- **لرؤية نتيجة التنفيذ اضغط على المفتاحين Alt+F5 تظهر النتيجة كما يلى :**

Welcome With CompuSience

- **للرجوع الى شاشة الكتابة اضغط أى مفتاح**

قواعدوأسس كتابة البرامج فى لغة ++C:

وإليك الإطار العام التالى الذى يعرض لأهم أسس وقواعد بناء البرامج فى لغة ++C :

```
#include <iostream.h>          // input/output commands
#include <conio.h>             // extra commands

#define constants              // constants for program

main()                         // start of the program
{
declaration of variables       // list all variables

   statement;             // Statements
   statement;             // More statements
   statement;             // as many as you want
```

تشتمل لغة ++C على عدد محدود جداً من المفاهيم الخاصة بها دون غيرها من اللغات الأخرى، ولكنها تشتمل فى مقابل ذلك على عدد كبير من الملفات الخارجية (external libraries). فبالإشارة إلى جزء الكود السابق - مثلاً - يتضح أنه قد بدأ بجملة # include التى تم بعدها كتابة أسماء كافة الـ libraries التى سيتم استخدامها فى البرنامج.

وعن مفهوم libraries هنا نجد أنه لا يختلف كثيراً عن مفهوم المكتبات العامة. فهى تشير فى لغة ++C إلى عدد من الملفات المشتملة على معلومات يستطيع المستخدم استدعائها فى أى وقت يشاء. تماماً مثل المكتبات العامة التى تزخر بالكتب والمعارف التى تظل رهنية الأرفف لحين الحاجة إليها واستخدامها. والإختلاف هنا أن هذه المكتبات يطلق عليها أسماء مثل : iostream.h و string.h. وهى libraries موجودة فى compiler لغة ++C. ويوجد غيرهما الكثير مما يحتوى على أوامر خاصة بعمليات مثل: الإدخال والإخراج والرسومات والتعامل مع بيانات strings.

وبالعودة إلى جزء الكود السابق مرة أخرى نجد أنه قد تم تعريف مجموعة من الثوابت (constants) بعد جملة define directive# وقبل بداية البرنامج الرئيسى. ويعنى الثابت هنا معلومة ما يمكن استخدامها فى أى وقت وأى جزء فى البرنامج، وتظل محتفظة بقيمتها ثابتة على مدار البرنامج كله.

وسنعرض فى الفقرات التالية لإطلالة سريعة على أهم مكونات البرنامج المكتوب بلغة ++C مع استمرار الإستعانة بالكود السابق. أما النموذج الكامل للبرنامج، فسيتم العرض له فيما بعد حتى يكون المستخدم قد تعرف على أنماط الكود الخاص بهذه اللغة وكيفية استخدامها.

تحتوى جميع البرامج المكتوبة بلغة ++C على كم هائل من الكود يعرف باسم Functions (مجموعة من الوظائف يؤديها البرنامج)، وسيتم العرض لها بالتفصيل فى الفصل الثانى عشر "قواعد functions". ولكن يكفى أن يعرف المستخدم الآن أنه لابد أن يحتوى أى برنامج مصمم بلغة ++C على Functions رئيسية اسمها main ( )، يتم من خلالها تنظيم جميع الأداءات فى البرنامج وهذا ما يتضح فى المثال السابق.

وبعد ذلك مباشرة تأتى علامة الحاصرة ({)، التى تعلن عن بداية البرنامج الرئيسى.

ويستطيع المستخدم بعد ها إدخال كافة المتغيرات التى سيستخدمها فى البرنامج. والمتغير عبارة عن معلومة ما قد تتغير فى خلال البرنامج. ولابد للمستخدم من أن يضع اسمًا لكل متغير منها فيما يعرف باسمidentifier (المحدد). وذلك فى عملية تعرف باسم declaring a variable (تعريف متغير). ويستطيع compiler بعد ذلك تحديد مكان لهذه المتغيرات فى الذاكرة تبعاً لاسم كل واحد منها. وستتضح هذه العملية فى الفصل الثالث، "أنواع البيانات وطرق التعامل معها فى لغة ++C".

وبعد عملية تعريف المتغيرات هذه، يقوم المستخدم بإدخال كافة الأوامر المراد تنفيذها في البرنامج في شكل جمل برمجية تستطيع لغة ++C التعامل معها. وتكمن أهمية البرنامج كله في هذه الجمل، وسيعرض الكتاب للعديد من التطبيقات الخاصة بها فيما بعد. وفي نهاية هذه الجمل البرمجية، لابد من وضع علامة الحاصرة ({) الأخرى لتحديد نهاية ( ) main كما هو موضح في المثال.

أما بالنسبة لقائمة الكلمات الإنجليزية التي تلي علامة (//)، والتي توجد على يمين الشاشة في البرنامج السابق، فهي عبارة عن مجموعة من التعليقات الغرض منها تيسير قراءة البرنامج. وبالتالي فهي لا تلعب أي دور في تنفيذ البرنامج غير أنها تعين على فهم الكيفية التي يعمل بها. بل وتعين المبرمج نفسه على فهم الكود إذا ما عاود مراجعته مرة ثانية بعد ذلك.

وقد توجد هذه التعليقات أيضاً في الأجزاء الأساسية من البرنامج لإزالة أي غموض أو لبس قد يواجه المستخدم عند قراءتها. وعلى ذلك، ليس من الضروري أن يضع المبرمج علامة التعليق (//) في كل سطر من الكود طالما أن التعليمات التي تؤديه واضحة للمستخدم.

وهناك طريقتان لإضافة التعليق في البرامج المكتوبة بلغة ++C:

(١) في حالة تعليق السطر الواحد، توضع علامة // بهذه الكيفية:

```
// This is a single line comment!
```

(٢) في حالة التعليقات الأكثر من السطر الواحد، توضع علامة /* and */ وبهذه الكيفية:

```
/* This is
a multiple
line comment! */
```

مفهوم "الثابت" في لغة ++C:

من الشائع في برمجة الكمبيوتر أن يتكرر الشيء أكثر من مرة في البرنامج. ولكن المستخدم للغة ++C يمكنه أن يضع اسماً قصيراً يشير إلى مجموعة البيانات المخزنة في ذاكرة الجهاز، واللازمة لتنفيذ البرنامج. ويعرف هذا بـ identifier (الاسم المحدد). وعلى ذلك، فكل ما يفعله المستخدم هنا لاستدعاء هذه البيانات من الذاكرة هو أن يذكر الاسم المحدد للمتغير الذي يحتوي عليها.

ويكون ذلك بدلاً من كتابتها على الشاشة فى كل مرة يحتاج فيها المستخدم إليها، ولا سيما وإن كانت هذه البيانات تمثل الجزء الأكبر من نص البرنامج. وتقوم لغة ++C باستبدال هذا الاسم بالجزء الذى يشير إليه نص البرنامج. ويستطيع المستخدم - فضلاً عن ذلك - إذا ما أراد إجراء أى تغيير على الجزء التكرارى فى النص أن يغيره مرة واحدة فقط عند إنشاء البرنامج، وليس فى كل مرة يستخدمه فيها.

ويوضح المثال التالى ذلك:

مثال (٢-٣): كيفية استخدام الرسائل المعرفة

```
#include <iostream.h>

#define message1 "Hello everyone...this is easy "
#define message2 "Look another line!"

main()
{
    cout << message1 << endl;
    cout << message2;
    cout << endl << endl << message1 << endl << endl;
    cout << message2 << endl;
    return(0);
}
```

- يستخدم أمر cout فى أى موضع فى البرنامج فى العملية الخاصة بعرض البيانات على الشاشة والمستخدم فيها أمر cout.

بعد أن تعرفنا على أساسيات كتابة البرامج في لغة C بصفة عامة يجب أن تأخذ خطوة أكبر نحو العمق لدراسة أساسيات اللغة نفسها ، من خلال هذا الفصل سنتعرض لدراسة مجموعة مختارة من البرامج فمن المعروف أن أي لغة برمجة يحكم عملها ثلاثة أمور أساسية.

- طريقتها في حفظ البيانات وتوصيفها.
- طريقتها في التعامل مع عمليات إدخال البيانات وإخراج النتائج.
- طريقتها في معالجة البيانات والمؤثرات التي تستخدمها.

وهو ما سنتحدث عنه في هذا الفصل ، بالطبع لن يكفي هذا الفصل لمناقشة هذه الموضوعات بالتفصيل لكن سنتناول منها ما يساعدنا على الفصول التالية ثم نستكمل التفاصيل مع باقي الكتاب

إطلاق الأسماء Identifiers

يطلق على أي اسم يستخدم في الإشارة إلى متغير أو دالة ... أو أي عنصر آخر في لغة C يقوم المستخدم بتوصيفه ، كلمة (معرف – Identifier) والمعرف هو ما يتم بواسطته تمييز عنصر عن آخر . ومع أغلب مترجمات اللغة المختلفة Compilers تستطيع كتابة معرف بطول من حرف – حتى 32 حرف أما مع Visual C++ فيمكنك كتابة معرف بطول 246 حرف ويشترط فيه.

1- أن يبدأ بحرف أو الشرطة السفلية (_) .

2- ألا يزيد عن الطول المسموح به.

3- ألا يكون إحدى الكلمات المفتاحية Key words .

4- ويجب ألا يشترك معرفان في نفس الاسم مطلقاً .

وإليك المثال التالي :

```
void main(void)
{
    int result;
        if ( result != 0 )
        printf( "Bad file handle\n" );
}
```

وقد استخدمنا فيه المعرف result للدلالة على اسم متغير من النوع Integer كما استعملنا main و printf كأسماء للدوال وكلها يطلق عليها مُعرف Identifier .

والأمثلة التالية توضح أسماء صحيحة وأخرى خاطئة:

السبب	الاسم
لا يبدأ الاسم برقم	1Star
غير مسموح باستخدام (!)	hi!tow
غير مسموح باستخدام (..)	high..balance
صحيح	star1
صحيح	high_balance

أنواع البيانات

كما تعرف أن البيانات التي نتعامل معها إما أرقام أو حروف أو كلمات و الارقام يمكن أن تكون صحيحة (أى ليس بها علامة عشرية) أو حقيقية أى بها علامة عشرية.

والحروف يمكن أن تكون حرف واحد أو اكثر من حرف وهكذا تختلف أنواع البيانات عن بعضها البعض و من الضرورى معرفة أنواع البيانات ومعرفة كيفية الاعلان عنها وكذلك كيفية استعمالها.

والجدول الاتى يوضح هذه الانواع وكذلك عدد البايت (Byte) التى يشغلها كل نوع

نوع المتغير	طوله بالبايت	المدى المسموح
حرف (char)	١	حرف أو رمز واحد
صحيح قصير (int)	٢	-٣٢٧٦٨ إلى ٣٢٧٦٨
صحيح طويل (long)	٤	-٢٠١٤٧٠٤٨٣٠٦٤٨ إلى ٢٠١٤٧٠٤٨٣٠٦٤٨
حقيقى (float)	٤	E-38 الى E+38
حقيقى مضاعف (double)	٨	E-308 الى E+308

ونوضح فيما يلى المقصود بكل هذه الانواع :

- متغير من نوع حرف أى متغير يصلح لتخزين حرف فقط
- متغير من نوع صحيح أى متغير يصلح لتخزين رقم صحيح (ليس به علامة عشرية مثل ٥ ،٥٧ ، ٥٤٤).

- متغير من نوع صحيح ولكن طويل (long) اى يستطيع أن يخزن رقم صحيح ضعف المتغير الصحيح العادى ونستعمل هذا النوع اذا كانت الارقام التى نتعامل معها أكبر من المساحة المخصصة للرقم الصحيح العادى والا سنحصل على نتائج خاطئة بالرغم من ان البرنامج سليم .
- متغير حقيقى أى متغير يصلح لتخزين رقم حقيقى (يقبل الكسور العشرية مثل ٣,٣ ٤٤,٥ ٣٣,٤)
- متغير حقيقى مضاعف اى يستطيع أن يخزن رقم حقيقى ضعف المتغير الحقيقى العادى

تسمية المتغير

- يخضع اسم المتغير لشروط معينة يجب أن تعرفها تجنبا لأخطاء قد تقع فيها وفيما يلى نوضح هذه الشروط :
- يجب ان يبدأ المتغير بحرف ثم يكمل المتغير بعد ذلك حروف أو أرقام ويجب ألا يحتوى على علامة خاصة سوى الشرطة التحتية (_).
- من الممكن أن يشتمل إسم المتغير حتى ٣٢ حرف وما زاد عن ذلك لا يلتفت إليه مترجم اللغة .
- يفرق المترجم بين الحروف الصغيرة و الكبيرة فالمتغير St يختلف عن المتغير st فاذا استعملا فى البرنامج يعتبرهما البرنامج متغيرين
- يجب ألا يكون المتغير باسم كلمة من الكلمات المحجوزة فى اللغة مثل int,return.

الإعلان عن المتغيرات

اذا كنت تستخدم مترجم للغة C/C++ فيمكن يتم الإعلان عن المتغيرات فى أى مكان بالبرنامج ولكن بشرط ان تكون قبل العبارات التى تستخدم هذا المتغير أما اذا كنت تستخدم مترجم للغة C فقط فيجب أن يكون الإعلان فى أول البرنامج لتلافى الأخطاء.

مثال :

```
int a;
float b;
```

شرح الدالة ( )Print F

هذه الدالة من أهم دوال C حيث تعد الوسيلة الأساسية لإظهار النتائج على الشاشة ، وفيما سبق - من خلال الفصل الأول - استخدمنا الدالة لطباعة نص معين ووضعناه بين علامتي التنصيص ، وفي الحقيقة أن printf() تستطيع أخذ أكثر من وسيطة argument لأداء وظيفتها عليها وهو ما يعطي مرونة أكبر للتحكم في النص الذي، تظهره والصورة العامة للدالة هي.

```
Printf(" نص بداخله محددات الشكل " , قيمة1 , قيمة2 , قيمة 3 )
```

. ولكي يتضح الأمر دعنا نطالع المثال التالي:

```
# include <stdio.h> //c احتواء ملف مصدر من مكتبات لغة
void main (void)// نقطة بداية البرنامج
{
printf( "my age is: %d \n",30);
 //استدعاء دالة من المكتبات
} //نقطة نهاية البرنامج
```

ولاحظ أننا أعطينا الدالة وسيطتين

- الأولى هي النص الموجود بين علامتي التنصيص my age is: %d ؛
- الثانية هي الرقم 30 .

ويفصل بينهما الفاصلة (,) ، وهو نفس الأسلوب الذي سيتبع دائماً في إعطاء وسيطات الدالة إذ دائماً تفصل بين الوسيطات بالفاصلة .

وتتعرف printf() على وسيطاتها على أساس أن الأولى هي نص والثانية التي تكون على يمين الفاصلة - حتى ولو كانت الأولى غير موجودة - هي القيمة Value وما تؤديه الدالة هو وضع القيمة إلى جوار النص وبالتشكيل Format الذي يحدده المحدد specifier الذي يمثله في هذا المثال %d والمحددات تخبر الدالة بأمرين

1 - أين تضع القيمة في النص.

2 - الشكل الذي تضعها عليه.

يمكننا أيضاً من خلال هذا المثال ملاحظة الرمز الجديد (\n) المستخدم في الوسيطة الأولى حيث يسبب هذا الرمز البدء في الطباعة في سطر جديد New Line وهو ما لا تفعله دالة printf() بنفسها إذ يجب أن يتم ذلك عمداً فالجملتين

```
printf("Hi, I did it.");
printf(" I am a programmer now");
```

تكون نتيجتها على الشاشة

```
Hi , I did it. I am a programmer now
```

بدون سطر جديد لأنه كما ذكرنا - الدالة لا تنشئ سطراً جديداً بمفردها

و المترجم لا يشعر بأي مسافات خالية ولا سطر جديد

الجدول التالي يوضح أنواع المحددات Spcifiers التي تستخدم لتعين موضع المتغير وشكله .

المحدد	يظهر المتغير في صورة
%C	حرف مفرد
%S	نص
%d	رقم صحيح بالنظام العشري Decimal Integer
%f	رقم به كسور عشرية
%e	رقم به أس العلمي
%g	رقم عشري عام يكتب على الصورة %f أو %e أيهما أقصر
%u	رقم صحيح بالنظام العشري وفي المدى 0-65535
%X	رقم صحيح بالنظام .HEX
%o	رقم صحيح بالنظام Octal
L	يستخدم مع الأرقام الصحيحة لجعلها Long (%0 , %4 , %d , %x)


تمرين

يمكنك الحصول على أحجام جميع أنواع البيانات وطباعتها باستخدام دالة printf() كالتالي:

```
printf(" size of integer: %d" , sizeof (int));
```

وهو ما يؤدي إلى طباعة الحجم الذي يشغله النوع int .

حاول كتابة برنامج يطبع أحجام جميع أنواع البيانات ولاحظ أن مثل هذا البرنامج تختلف نتائجه على حسب نظام التشغيل الذي يعمل عليه الحاسب خاصة مع النوع int .

هذا وتتواجد بعض المؤثرات الأخرى مثل النقطة (.) أو السهم (<-) والذي يتكون من علامتي - ، > وتستخدم هذه المؤثرات مع المؤشرات والتركيبات وسيلي مناقشتها في حينه.

المؤثرات Operators

تُعد لغة C من أغنى لغات البرمجة في استخدامها للمؤثرات . ونعني بالمؤثر رمز يستخدم لإخبار المترجم Compiler أن يؤدي مهمة معينة ، قد تكون حسابية أو منطقية . ويتواجد في C ثلاثة أنواع أساسية من المؤثرات:

- رياضية
- منطقية
- مؤثرات خاصة

1 - المؤثرات الرياضية Arithmetic

وبالتأكيد تعرفت على المؤثرات الرياضية + ، -، * ، / فهي تستخدم في لغة C كما في أي لغة أخرى ، والجدول التالي يوضح المؤثرات الرياضية واستخداماتها:

المؤثر	استخدامه
-	الطرح
+	الجمع
*	الضرب
/	القسمة
--	Decrement – التناقص
++	Increment – التزايد
%	Modulus Division – الباقي

وبالطبع تعرف الأربعة الأولى منها لكن قد يكون المؤثر % جديداً عليك وهو باختصار ينتج باقي القسمة بمعنى أن 7%2 سيكون ناتجها = 1 ويسمى معامل الباقي.

أما المؤثران -- و ++ فهما ربما يكونان جديدين بالنسبة لك وهي تستخدم ببساطة في عمليات التناقص والتزايد لأحد المتغيرات بمقدار الوحدة وربما لا يوجد لهما مثيل في أي لغة أخرى وهي تستخدم كالتالي:

```
X=X+1;
```

لها نفس النتيجة من الجملة

```
++X  ;
or
X++  ;
```

وفي حالة مؤثر التناقص فإن

```
X=X-1;
```

وتؤدي وظيفتها الجملة

```
--X;
or
 X--;
```

وبصفة عامة لا فارق في حالة استخدام --X أو X-- كجمل مستقلة وكذلك بالنسبة للإضافة ++X أو X++ ، لكن في حالة الاستخدام داخل تعبير رياضي Expression فإن الفارق كبير عندما نستخدم X-- فإن القيمة التي تدخل في الحساب هي قيمة X بعد التناقص ، أما في حالة استخدام --X فإن القيمة المستخدمة في الحساب هي قيمة X قبل التناقص وإليك المثال التالي:

```
X=15
Y=++X;
```

في هذه الحالة تصبح قيمة Y = 16

```
X=15;
Y=X++;
```

في هذه الحالة Y = 15 .

لأنها تأخذ قيمة X قبل حدوث الزيادة .

في جميع الحالات فإن X ستصبح 16 ، ولكن توقيت حدوث الزيادة هو الذي يختلف في الحالتين.

وتتم العمليات الحسابية بنفس الأسلوب المستخدم في جميع اللغات من حيث أولويات التنفيذ ، فيتم التنفيذ من اليسار إلى اليمين مع إعطاء أولوية للضرب والقسمة على الجمع والطرح ما لم يكن هناك أقواس تحدد العملية وتغير الترتيب.

ويجب أن تلاحظ أن هذه المؤثرات تعمل مع جميع أنواع البيانات المعروفة في C لكنها تؤدي إلى اختصار ما يناسب نوع البيانات فعلى سبيل المثال المؤثر / عندما يستخدم مع integer يؤدي إلى حذف باقي نواتج القسمة مثلاً

```
X= 9/4
```

سيؤدي إلى قيمة X=2 ويحذف الباقي إذا كانت X معرفة على أنها Integer .

2 - المؤثرات المنطقية والعلاقات

وهي المؤثرات التي تستخدم في اختبار صحة علاقة قيمة بقيمة أخرى ، ويبينها الجدول التالي:

م	المؤثر	معناه
1	>	أكبر من
2	>=	أكبر من أو يساوي
3	<	أصغر من
4	<=	أصغر من أو يساوي
5	==	يساوي
6	!=	لا يساوي
7	&&	و - And
8	!	ليس - Not
9	||	أو - OR

والمؤثرات الأربعة الأولى بالتأكيد سبق لك التعامل معها لكن المؤثر الخامس يختلف عن باقي لغات البرمجة ويستخدم بهذه الصورة للتفرقة بينه وبين مؤثر التخصيص (=) وهو يختبر تساوي القيمتين على جانبيه.

- المؤثر =! يختبر عدم تساوي القيم على جانبيه.
- المؤثر && يختبر صحة الشرطين ويكون صحيحاً في حالة وحيدة وهي صحة كلا الشرطين على جانبيه فمثلاً.

```
X>5 && Y < 3 ;
```

وبذلك يجب أن تكون X>5 وفي نفس الوقت Y<3 حتى يصبح الشرط صحيحاً.

- المؤثر ! ينتج قيمة صحيحة فقط في حالة عدم تحقق الشرط.
- المؤثر || ينتج قيمة صحيحة في حالة تحقق أي من الشرطين أو كلاهما في نفس الوقت.

وفي جميع حالات استخدام هذه العلاقات فإن تحقق العلاقة ينتج عنه قيمة 1 أي أن الشرط صحيح True وعدم تحققها ينتج قيمة صفر أي أن الشرط خطأ false والجدير بالذكر أن جميع المؤثرات السابقة تـأتي في ترتيب أولويات الحاسب بعد المؤثرات الحسابية فمثلاً

```
15 > 7*2
```

ولا فارق بينها وبين

```
15>(7*2)
```

في الحالتين سينتج قيمة صحيحة = 1

3 - بعض المؤثرات الخاصة

أ) المؤثر ؟

يستخدم هذا المؤثر كبديل عن If ... else في حالات كالتالي

```
X=10
if(x>8) Y=100;
Else Y=200;
```

هذه الجمل يمكن اختصارها في الجملة التالية

```
X=10
Y=X>8 ? 100:200;
```

وهي باختصار تؤدي إلى اختبار الشرط الأول فإن تحقق يتم إعطاء Y قيمة التعبير الذي يلي ؟ مباشرة

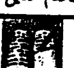
فإن لم يتحقق تأخذ Y قيمة التعبير الموجود بعد : وصورتها العامة

```
Exp1 ? Exp2 : Exp3
```

حيث Exp1 هو الشرط أو الاختبار

ب) مؤثرات & ، *

يستخدم هذين المؤثرين في حالات استخدام مؤشر Pointer للتعامل مع عنوان أحد المتغيرات في الذاكرة.



سيأتي شرح المؤشرات في الفصل السابع

وببساطة يمكن عمل مؤشر لأي متغير هذا المؤشر يحتوي على عنوان المتغير أو مكانه في ذاكرة الحاسب ، فمثلاً

```
S=& Count
```

هذا يعني أن يتم وضع عنوان موضع المتغير Count في المتغير S فعلى سبيل المثال لو أن المتغير Count لـه قيمة 77 وهو موجود في العنوان 1000 من الذاكرة فإن محتويات S تصبح 1000

وهو بذلك لا يؤثر بحال من الأحوال على قيمة Count .

ويعتبر المؤثر * هو العكس من & حيث يستخدم للإشارة إلى موضع في الذاكرة للحصول على قيمة المتغير الموجود فيه فمثلاً

```
R=*S
```

ينتج عنها وفقاً للأرقام في المثال السابق أن تصبح قيمة R = 77 حيث هي القيمة التي يشير إليها العنـــوان الموجود في S .

ج) المؤثر *Sizeof ( )*

تعتبر معظم مترجمات لغة C الكلمة المفتاحية Sizeof مؤثر يعيد قيمة تعبر عن طول أو حجم الوسيطة التي يأخذها في الذاكرة فمثلاً

```
sizeof(Z);
```

ينتج عنها حساب حجم الذاكرة الذي يشغله المتغير Z .

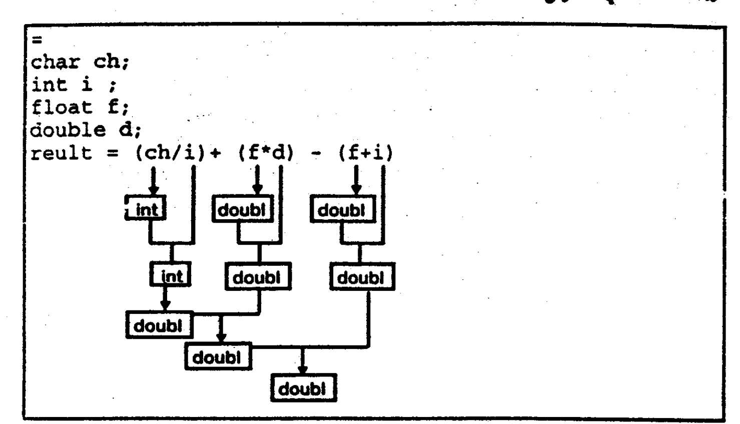
التعبيرات

المؤثرات والمتغيرات والثوابت تستخدم كلها من خلال تعبيرات رياضية Expressions ... وهي في لغة C كما في معظم لغات الحاسب تتبع قواعد الجبر العادية لذلك فليس هناك جديد غير أننا نود التنبيه إلى عـــدة نقاط.

1 - عندما تكتب أي تعبير رياضي تستخدم فيه أكثر من نوع بيانات فإنه يتم تحويلها إلى نفس النـــوع - ويقوم المترجم بتحويل جميع الأنواع إلى النوع الأعلى وفقاً للقواعد التالية:

- جميع Char. يتم تحويلها إلى integer وتتم عليها الحسابات وفقاً لذلك كما أن جميع Float تتحـــول إلى Double .
- عند وجود عملية على نوعين أحدهما Double يتم تحويل الأخر إلى Double.

وإليك المثال التالي للتحويل



2 - عند الحاجة إلى التحكم في الناتج تستطيع استخدام الجملة على الصورة

$$\text{int } \frac{x}{2}$$

مما يجعل الناتج دائماً من النوع integer مهما كان نوع x .

فمثلاً الأسطر التالية

```
#include <stdio.h>
main(void)
{
int i ;
for(i=1; i<=100 ; ++i)
printf("%d/2 is: %f", i , (float) i/2) ;
}
```

ويتضح منها أننا أعلنا عن i في صورة متغير integer وعند دخول الدوارة تأخذ في طباعة قيمة i من 1 - 100 وتضع بعدها قيمة i/2 ثم تظهر إلى جوارها صورة i في شكل Float .

في الحالة العادية عند طباعة الأرقام الفردية ستكون هناك مشكلة لأنها ستطبع integer مما يؤدي إلى حذف الكسر لكن وضع التعبير (float)i/s في الجزء الثالث من دوارة for يضطر الحاسب إلى طباعة القيمة في الصورة float .

3 - يمكنك استخدام بعض الاختصارات في كتابة التعبيرات في لغة C فكلا التعبيرين التالين متساوي

```
x = x+5 ;
x+=5 ;
```

والصورة العامة

(تعبير رياضي) (مؤثر) نفس المتغير = متغير

يكافئ

التعبير الرياضي = (المؤثر) المتغير

أي أن

```
d = d -(100*y)
d - = 100 *y
```

ويجب أن تعتاد على هذا الأسلوب لأنك ستجده بكثرة في العديد من برامج C المتخصصة.

نظرية Operator or Precdence (أولوية المعامل):

منذ حوالى أكثر من أربعة آلاف عام واجه اليونانيون القدماء مشكلة فى تحديد الترتيب الصحيح لإجراء العلميات الحسابية المختلفة بالشكل الذي يضمن الحصول على الناتج الصحيح فى النهاية. ولحل هذه المشكلة، قام اليتانيون بوضع مجموعة من الأسس والقواعد الخاصة بعلم الحساب، واطلقوا عليهـا نـظريـة Operator Precdence (أولوية المعامل)، ولا نزال نستخدم هذه النظرية حتى يومنا هذا. والفكرة أن هناك نوعان من العمليات الحسابية: عمليات ذات أولوية كبيرة (High-precdence) وعمليات ذات أولوية أقل (low-precdence).

وتفصيلاً لذلك: هناك عمليات حسابية لابد من تنفيذها قبل غيرها داخل العملية الحسابية الواحدة. وتعد عمليتى الضرب والقسمة من النوع الأول، بينما الطرح والجمع من النوع الثانى. ويكون ترتيب الأولوية على هذا النحو:

أولاً: الضرب (*) والقسمة (/)

ثانياً: الجمع (+) والطرح (-)

وعلى ذلك، فلو اشتملت إحدى العمليات الحسابية على جميع هذه المعاملات معًا، فإن عمليتى الضرب والقسمة يكون لهما الأولوية فى التنفيذ قبل عملتى الجمع والطرح. ولكن إذا كانت العملية لا تشتمل إلا على النوع الأول فقط (الضرب والقسمة)، فليس ثمة أى فرق فى تنفيذ إحداهما قبل الأخرى. وبالمثل، لو اشتملت العملية الواحدة على النوع الثانى فقط (الجمع والطرح). ومعنى ذلك أنه لا توجد أولوية فى التنفيذ بالنسبة لعمليات النوع الواحد. ويوضح المثال التالى ذلك:

لو قلنا أن

A + B * C

ثم عوضنا بعد ذلك عن هذه الحروف، التى ترمز بالطبع إلى متغيرات فى البرنامج بالقيم التالية:

A = 3

B = 2

C = 4

إذا A + B * C

ستصبح 3 + 2* 4

وستساوى 11

ولابد للحصول على هذا الناتج الصحيح أن يتم ضرب قيمة المتغير B فى قيمة المتغير C أولاً، ثم يتم إضافة قيمة المتغير A إلى الناتج. ولو حدث العكس، فجمعت A وB أولاً، ثم ضرب الناتج فى C، فسيكون الناتج خطأ: 20.

ما هى علامات prantheses وفيم تستخدم؟

علامات prantheses عبارة عن قوسين متقابلين ( )، يؤديان وظيفة هامة فى معالجة الأعداد فى لغة ++C. وتتمثل الوظيفة الأساسية لهما فى تغيير ترتيب أداء عمليات الحسابية. فلو وضعنا- مثلاً- عملية الجمع التى جاءت فى المثال السابق بين قوسين بهذا الشكل:

(A + B) * C

فسيتغير الناتج تماماً.

وبناءاً على هذا التغيير الذى حدث فى تركيب هذه الجملة الحسابية، فسيتغير مدلول تنفيذها بالنسبة للغة ++C: سيفهم الكمبيوتر أن قيمة المتغير A قد أضيفت إلى قيمة المتغير B أولاً، وضرب الناتج بعد ذلك فى قيمة المتغير C. وبذلك يكون الناتج الإجمالى للعملية الحسابية كلها هو 20 وليس 11. وبالإضافة إلى ذلك، فقد تستخدم prantheses أيضاً فى أى جزء من نص البرنامج لإزالة أى لبس أو غموض قد يعتريه. ومن ثم تكون إمكانية قراءة البرنامج أكبر وأيسر.

ولاحظ أنه من الممكن بعد استخدام prantheses الحصول على ناتج واحد من صيغتين مختلفتين، بهذا الشكل:

A + B * C

A + (B * C)

ولكن مع فارق واحد وهو: أن الصيغة الثانية ستكون أكثر وضوحاً للمستخدم.

مثال تطبيقى على معالجة البيانات بهذه المعاملات الحسابية فى لغة ++C:

والآن حاول تشغيل البرنامج الموجود فى مثال (٤-١) التالى. ولكن، حاول أن تتنبأ أولاً بالنتائج الثلاثة المتوقعة. وقارن بعد ذلك بين توقعاتك وما سيظهر أمامك على الشاشة. وإن صحت هذه التوقعات، فتأكد أنك قد فهمت الفكرة.

مثال (٤-١): بيان لكيفية تنفيذ عمليات الجمع والطرح والضرب فى لغة ++C:

```
#include <iostream.h>

main()
{
  int A = 12;
  int B = 22;
  int C;
  int D;
  int E;

  C = A + B;                    // ADD
  D = A - B;                    // SUBTRACT
  E = A * B;                    // MULTIPLY
  cout << "ADD  : " << C << endl;
  cout << "SUB  : " << D << endl;
  cout << "MULT : " << E << endl;
  return(0);
}
```

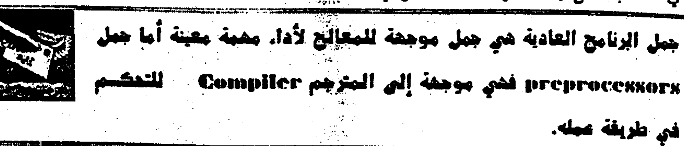
الموجهات Directives

تتمتع لغة C بخاصية لا تتوافر في لغات أخرى عديدة من لغات المستوى الرفيع High Level Languages هذه الخاصية تتمثل في إمكانية وضع جمل في بداية الملف تعتبر ملاحظات أو إرشادات للمترجم Compiler لأداء بعض العمليات قبل البدء في ترجمة ملفات المصدر ولذلك تسمى هذه الجمل جمل ما قبل المعالجة Pre processors .

وحتى ندرك هذه النقطة دعنا نراجع الخطوات التي يقوم بها المترجم عند كتابة سطر كالتالي في أحد البرامج.

```
Score = 15 ;
```

من خلال هذا السطر نطلب من المترجم أن يوجد تعليمات لغة الآلة التي يمكن تنفيذها بواسطة شريحة المعالج في الحاسب لجعل قيمة المتغير Score السابق الإعلان عنه = 15.



جمل البرنامج العادية هي جمل موجهة للمعالج لأداء مهمة معينة أما جمل preprocessors فهي موجهة إلى المترجم Compiler للتحكم في طريقة عمله.

ويتواجد مع لغة C عدة جمل لتوجيه المترجم أشهرها جملتي

```
# include < ملف رأس >
# define قيمة متغير
```



جمل التوجيه تبدأ دائماً بالرمز # ويمكن وضعها في أي مكان من ملف المصدر إلا أنه مصطلح عليه بين مستخدمي لغة C أن توضع في بداية الملف قبل بدء دالة main ( ) أو قبل أي دالة مرتبطة باستخدام الموجه.

الموجه Define

الاستخدام الشهير لجملة define #هو إعطاء اسم بقيمة الثوابت مثل تسمية الرقم 3.14159 باسم PI فإن جملة define# تستبدل عبارة بأخرى ، وإليك المثال التالي:

```
/* circle.cpp */
/* برنامج لحساب مساحة دائرة باستخدام الموجه لتحديد قيمة
الرقم الثابت pi */
# include < stdio.h >
# define pi 3.14159
float area (float);          /* الشكل المبدئي */
void main (void)
{
float radius ;
printf (" enter radius of circle :") ;
scanf ("%f" , &radius );
printf  ("  area  of  circle  is  :  %2f  \n "  , area
(radius));
printf ( "pi in this case =  %f \n " , pi);
}
/* area ( ) */
```

ولعلك تتساءل ما هي فائدة هذا الموجه ؟؟!

ألم نكن نستطيع استخدام القيمة المستبدلة مباشرة بدلاً من تسميتها واستخدام الاسم ثم جعل المترجم يستبدل الاسم بالقيمة ؟؟!

أولاً : من خلال هذا الأسلوب بجعلنا في جعل البرنامج يمكن قراءته بشكل اسهل لأنه في الغالب قراءة كلمة pi أكثر توضيحاً من وجود الرقم 3.14159 وربما تكون الحالة السابقة رقم مشهور ... لكن هناك من الحالات العديدة لن تكون الثوابت المعبر عنها بهذه الشهرة.

ثانياً : عند الرغبة في التعديل لجعل الثابت مثلاً 3.141596 بزيادة رقم عشري للحصول على دقة أعلى ، فإن ما يحتاج إلى تعديل هو فقط جملة الموجه وبعدها سيتم التعديل في كل الملف تلقائياً.

والسؤال الثاني ... ألم نكن نستطيع استخدام متغير نضع به القيمة الثابتة وبالتالي نستخدم في باقي الملف اسم المتغير ويتم بعدها معاملته بشكل عادي كباقي المتغيرات بدلاً وضع طريقة جديدة ؟؟!

بالطبع كان يمكن ذلك خاصة أنه يحقق الفائدتين السابقتين حيث يمكن إعطاء المتغير اسم ذو معنى ... وعند تغير قيمته في بداية البرنامج سيتم تغيرها في جميع المواضع.

لكن هناك ثلاثة جوانب تجعل هذه الخطوة غير مرغوب فيها.

أ- الملفات التي يستخدم فيها الموجه تكون أسرع كثيراً في عمليات المعالجة عن تلك التي يستخدم فيها متغير لأداء هذه المهمة.

ب - أنه ليس من التفكير المنطقي السليم أن تصنع متغيراً لا تتغير قيمته فمادامت هذه القيمة ثابته فلتكن ثابت لا علاقة له بالمتغيرات.

ج - يتواجد دائماً احتمال حدوث تغير غير مقصود في قيمة المتغير أثناء تنفيذ البرنامج مما يفقد هذا الثابت قيمته ، لذلك فإننا نقلل احتمالات الخطأ عند استخدام جملة #defin .



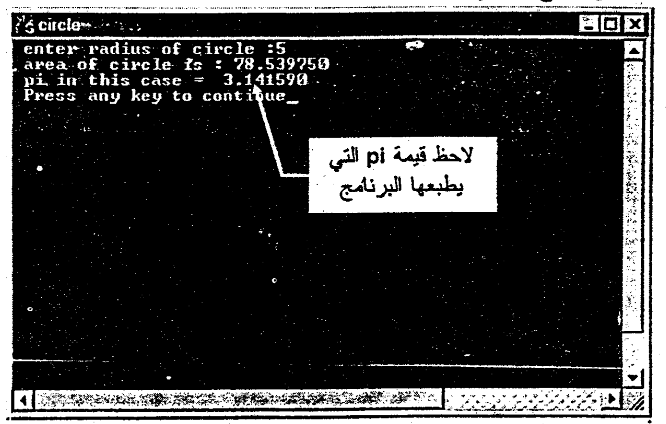
أدخلت قياسات ANSI الجملة Const للإعلان عن متغير لا يمكن تغيير قيمته على الصورة

Const float pi = 3.14159

وهي تحقق معظم الفوائد الممكن الحصول عليها عند استخدام #define وبالفعل انتشرت هذه الطريقة كبديل عن الأسلوب القديم.

```
/* function to calculate area */
float area (float rad )
{
return (pi*rad*rad); //إعادة المساحة إلى البرنامج الرئيسي
}
```

وعند تشغيل البرنامج ستحصل على

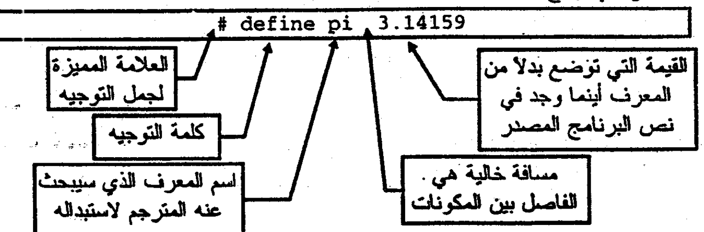


وما حدث بالضبط أنه عند ترجمة هذا الملف فإن المترجم في البداية يتعرف على السطر

```
# define pi 3.14159
```

وذلك على اعتبار أنه يبدأ بالعلامة # وبعدها فإنه يمر خلال البرنامج ويستبدل كل كلمة pi بالرقم 3.14159 على طريقة الاستبدال التي تحدث في برامج معالجة النصوص word processors .

والشكل التالي يوضح كيفية الاستبدال



الموجه #include

هذا الموجه directive هو واحد من أهم موجهات لغة C ويتم عن طريقة جعل أحـــد ملفـــات المصـــدر Source files محتوى داخل ملف مصدر آخر ،وهو ما يتيح الاستفادة سواء بمكتبات لغـــة C أو حـــتى برامج فرعية تكون كتبتها بنفسك من قبل.

وبذلك فهو يقلل إعادة كتابة كود أي خطوة عدة مرات ، ويتم كتابة #include بطريقـــة مـــن اثنتـــين لتعريفها باسم الملف المطلوب إحضاره ، وموضحه مثل

```
#include "search.h"
```

وهي تجعل المترجم يبحث عن الملف search.h في المجلد أو الفهرس المحتوى على نفس ملف المصدر الـــذي يحتويه.

والشكل الثاني لكتابة جملة #include هو

```
#include <search.h>
```

وهو يجعل عملية البحث عن الملف search.h تتم في الفهرس القياسي الذي يتم تحديده لهذا الغرض بواسطة عملية تثبيت البرنامج setup .

ويسمى هذا الفهرس standard header directory .

ملفات الرأس Header Files

تأتي لغة C مع مجموعة قياسية من الملفات التي يطلق عليها Header Files هذه الملفات تحتوي على عدد من الدوال والبرامج الفرعية التي تصلح للاستخدام داخل أي برنامج وعادة يتم وضـــع هـــذه الملفـــات في الفهرس الفرعي include\ داخل الفهرس الذي قمت بتهيئه Visual C++ بداخله ،؟ وبالفعل استخدمنا في بعض البرامج السابقة الملفات Stdio.h من داخل هذا الفهرس.

وسيلي مناقشة هذه المجموعة من الملفات فيما بعد لكن مؤقتاً يتواجد داخل هذا الفهرس جميع الملفات الـــتي تمثل مكتبات الدوال في لغة C .

وتسمى هذه الملفات بملفات الرأس header files لأنـــها توضع في بداية ملف المصدر ويتم عن طريقـــها تحديد الدوال التي ستستخدم داخل البرنامج لأن العملية التي تتم بواسطة استخدام جملة #include تشبـــه عملية تحديد الشكل المبدئي prototype للدوال المكتوبة داخل ملف المصدر.

وبالتالي تصبح كأنك قمت بعمل الشكل المبدئي لهذه الدوال باستخدام جملة include# والجدول التـــالي يوضح أهم ملفات الرأس الموجودة مع مكتبات لغة C والدوال التي توفرها.

اسم الملف	نوعية الدوال التي يحتويها	ملاحظات
alloc.h	Dynamic memory allocation	الدوال التى تتعامل مع الذاكرة .
assert.h	Defines assert()	تعريف الدالة assert .
bios.h	BIOS interface functions	تعريف دوال واجهة التعامل مع BIOS .
conio.h	Direct console I/O Function	تعريف دوال التعامل مع لوحة المفاتيح والشاشة.
ctytpe.h	Character-related functions	تعريف الدوال المرتبطة بالرموز.
dir.h	Directory-related functions	تعريف الدوال المرتبطة بالمجلدات.
dos.h	DOS interface functions	تعريف الدوال المرتبطة بنظام التشغيل DOS
errno.h	defines various error codes	التعامل مع رسائل الخطأ المختلفة.
fcnt.h	defines various constants used by the UNIX-like file system	تعريف ثوابت متعددة تستخدم بواسطة نظام الإدخال والإخراج شبيه UNIX .
float.h	defines floating point limits	تعريف حدود الأرقام العشرية float .
Graphics.h	graphic-related functions	الدوال المرتبطة بالرسومات
io.h	Low-level I/O functions	دوال المستوى الأدنى للإدخال والإخراج.
Limits.h	defines various integer limits	تعريف حدود المختلفة للأرقام الصحيحة.
Locale.h	Country-specific	الدوال المرتبطة بالدول

ملاحظات	نوعية الدوال التي يحتويها	اسم الملف
	functions	
تعريف الدوال الرياضية.	mathematical functions	math.h
تعريف دوال الذاكرة	Memory manipulation functions	mem.h
دوال التحكم في المعالجة	process control functions	process.h
تحتوي على تعريف الدوال setjmp() و longjmp()	required by setjmp() and longjmp()	setjmp.h
دعم مشاركة الملفات.	support for file-sharing	share.h
تعريف دوال signal() و raise()	support for signal() and raise()	signal.h
تعريف أنواع البيانات الرئيسية.	defines standard types and macros	stddef.h
دوال الإدخال والإخراج القياسية	standard I/O function	stdio.h
تعريفات لدوال متنوعة	miscellaneous functions	stdlib.h
الدوال المرتبطة و بالنص	string-related function	tring.h
الثوابت المرتبطة بالملفات	file-related constants	sys\stat.h
تعريف الدالة ftime ()	supports the ftime() function	sys\timeb.h
تعريف الدوال المرتبطة بالوقت والتاريخ.	Time-and date-related functions	time.h
تعريف أنواع من الثوابت الشهيرة	Various	valuw.h

ملاحظات	نوعية الدوال التي يحتويها	اسم الملف
كثيرة الاستخدام.	implementation-dependent constants	


الفصل الثالث
دوال الإدخال
والإخراج

الفصل الثالث

دوال الإدخال والإخراج

المزيد عن استخدام أمر cout مع المتغيرات:

ستظهر على الشاشة الرسالة : 42 بعد تشغيل البرنامج وبناؤه . وبذلك يمكن تلخيص خطوات التنفيذ فى النقاط التالية :

- حددت جملة ;int number أن البيانات فى المتغير number من نوع integer .
- وضعت جملة ;number = 42 قيمة داخل المتغير .
- عرضت جملة ;cout << number القيمة الموجودة فى المتغير على الشاشة .

وننتقل الآن إلى نوع جديد من أنواع البيانات هو : characters وسيتم التعامل معه بنفس الكيفية السابقة .

مثال (٣-٢): كيفية عرض محتويات متغير من نوع char على الشاشة:

```
#include <iostream.h>

main()
{
        char letter;        // set up a character variable

        letter = 'C';       // give it a value
        cout << letter;      // display to screen
        return(0);
```

اكتب البرنامج، واحفظه، ثم أجرى له عملية compile، وابنيه . وعند تشغيله، سيظهر على الشاشة حرف c.

ويسرى نفس الشئ على البيانات من نوع float .

مثال (٣-٣): كيفية عرض محتويات متغير من نوع Float على الشاشة:

```
#include <iostream.h>
```

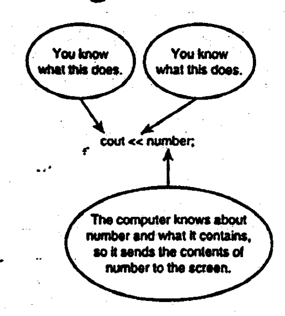
```
main()

        float number2;         // set up a float variable

        number2 = 98.6;        // give it a value
        cout << number2;       // display to screen
        return(0);
```

اكـتـب البـرنامج، واحـفظه، وأجـرى له عـمليـة Compilation، وابنيـه. وعند تشغيله، سيظهر على الشاشة القيمة التالية: 98.6.

وهكذا يتضح من الأمثلة السابقة أهم الأسس والقواعد التى يقوم عليها فكر لغة C++. ويتضح أيضاً أنها لغة سهلة جداً سواء فى التعلم أو الاستخدام، وذلك على الرغم من أنها من لغات البرمجة الحديثة الأكثر تطوراً. والآن، نعرض لبعض الصيغ المختصرة الخاصة بلغة C++ التى تذكر أول ما نذكره منها أن المستخدم يمكنه أن يكتب اسم المتغير والقيمة الخاصة به فى سطر واحد فقط كما هو موضح فى الشكل التالى:

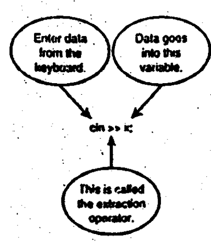


كيفية تعريف المتغير وتكوينه فى خطوة واحدة

أمر cin، ووظيفته فى البرنامج:

من المعروف أن المعلومات الداخـلة إلى المتغـيرات يتم تحديدهـا فى (compile time)؛ وفيه يقوم المستخدم بكتابة قيم المتغيرات فى source code ثم يجرى له عملية compilation. وبذلك، ستظل جميع القيم ثابتة على مدار البرنامج. وفى حالة ما إذا أراد المستخدم إدخال قيم أخرى مختلفة، فإنه لابد له من إعادة كتابة source code مرة أخرى. لذا، يفضل دائماً أن يدخل المستخدم هذه القيم أثناء تشغيل البرنامج (فيما يعرف باسم runtime). وبذلك سيكون المستخدم فى غير حاجة إلى كتابة الكود مرة ثانية لتغيير

القيم الموجودة فيه . ويكون ذلك باستخدام أمر cin . ولو فرضنا أن اسم المتغير x ، فإن صيغة الأمر هي :



تنفيذ أمر cin

إدخال بيانات إلى المتغير:

يوضح مثال (٤-٦) الكيفية التي يستخدم بها أمر cin لجمع المعلومات وإدخالها إلى المتغير . والبرنامج المعروض هنا هو نفسه البرنامج الذي سبق عرضه في مثال (٤-١) السابق ، ولكن بعد تعديله لإدخال البيانات أثناء تشغيل البرنامج . وفيه يطلب من المستخدم أن يدخل كافة المعلومات المطلوبة باستخدام أمر cin . وذلك بعد تحديد اسم المتغير المراد إدخال المعلومات إليه . ويزيد هذا الأسلوب من تنفيذ أداء عملية الإدخال عما لو تم استخدام قيم ثابتة فيه . وعلى المستخدم أن يحاول استخدام قيم مختلفة ومعقدة مع ملاحظة النتائج في كل مرة .



ماهي استخدامات ملفات iostream.h؟

الآن وبعد التعرف على أوامر cin و cout، نريد أن نوجه عناية القارئ إلى أن هذه الأوامر هي نفسها أوامر عمليات الإدخال والإخراج المعروفة، ولكن بلغة ++C. وهي مجموعة من الأوامر التي توجد في iostream.h library. ويتم استدعائها للاستخدام في البرنامج بوضع الجملة البرمجية التالية في بداية البرنامج الرئيسي على أنها library header الملفات الداخله فيه:

```
#include <iostream.h>;
```

مثال (٤-٦): بيان لكيفية تنفيذ أمر cin مع مزيد من التوضيح لعمليات الجمع والطرح والقسمة:

```
#include <iostream.h>

main()
{
   int A;
   int B;
   int C;
   int D;
   int E;

   cout << )Enter the value for A : );
   cin >> A;
   cout << )Enter the value for B : );
   cin >> B;
   C = A + B;              // ADD
   D = A - B;              // SUBTRACT
   E = A * B;              // MULTIPLY
   cout << "ADD  : " << C << endl;
   cout << "SUB  :  " << D << endl;
   cout << "MULT : " << E << endl;
   return(0);
}
```

أهم الدروس المستفادة:

- مفهوم نظرية operator precedence (أولوية المعامل) التى يكون لعمليتى الضرب والقسمة فيها الأولوية فى التنفيذ قبل عمليتى الجمع والطرح.

يشار لعمليات الجمع فى لغة ++C بالمعامل (+) . وقد يأتى الناتج محدداً لمتغير ما :

```
answer = first + second;
```

يشار لعمليات الطرح فى لغة ++C بالمعامل (-) . وقد يأتى الناتج محدداً لمتغير ما :

```
answer = first - second;
```

يشار لعمليات الضرب (*) . وقد يأتى الناتج محدداً لمتغير ما :

```
answer = first * second;
```

يشار لعمليات القسمة بالمعامل (/) . وقد يأتى الناتج محدداً لمتغير ما :

```
answer = first / second;
```

Parnthescs عبارة عن قوسين متقابلين ( ) يستخدمان فى لغة ++C لتحديد ترتيب تنفيذ خطوات العملية الحسابية، والمساعدة على فهمها وقراءتها بعد ذلك .

يستخدم أمر cast لتحويل البيانات فى المتغير من نوع لآخر . ويكون ذلك لمدة محددة هى مدة كتابة سطر الكود، ثم تعود البيانات إلى النوع الأصلى بعد ذلك .

تأخذ الجملة البرمجية الخاصة بتنفيذ عملية التحويل (casting) التركيب التالى :

```
answer = (floot) first / second;
```

يستخدم أمر cin لإدخال قيمة إلى حد المتغيرات فى البرنامج، وصغتيه هى : cin>> first

دوال إدخال حرف واحد getchar(),getche(),getch()

بالرغم من أن الدالـة scanf() تستقبل جميع أنواع البيانات الا ان لغة C تشتمل على دوال اخرى تتعامل مع أنواع خاصة من البيانات كالحروف و العبارات الحرفية ونوضح فيما يلى أهم هذه الدوال.

الدالة getchar()

ملف التوجيه : *stdio.h*

تستخدم لإدخال حرف واحد ويظهر الحرف على الشاشة بعد الكتابة ولاتسمح بالانتقال الى الامر التالى الا اذا ضغط المستخدم مفتاح الادخال (Enter) و هذا يشبه ما يحدث مع أمر Format فى نظام التشغيل DOS حيث يعطيك أمر Format الرسالة Format Another (y/n) وينتظر لتكتب حرف Y أو حرف N ويظهر الحرف وينتظر حتى تضغط على مفتاح الادخال Enter.

مثال :

```
char a;
a=getchar();
printf("%c",a);
```

الدالة ***getche()***

ملف التوجيه : **conio.h**

تستخدم لإدخال حرف واحد ويظهر هذا الحرف على الشاشة ولكنها تختلف عن الدالة getchar() فى أنها لاتحتاج الى الضغط على مفتاح الادخال(Enter) للانتقال للسطر التالى وتعمل هذه الدالة بطريقة مشابهة. لرسالة الخطأ التى تظهر فى نظام التشغيل بهذا الشكل

(A)bort, (R)etry, (F)ail ?

حيث يتم التنفيذ بمجرد الضغط على أحد الحروف A,R,F بدون حاجة لضغط مفتاح الإدخال ويظهر الحرف على الشاشة

مثال :

```
char a;
a=getche();
printf("%c",a);
```

الدالة getch ()

ملف التوجيه : conio.h

تستخدم لإدخال حرف واحد ولكن تختلف عن الدالتين السابقتين فى أن هذا الحرف لا يظهر على الشاشة وكذلك فى أنها لاتحتاج الى الضغط على مفتاح الادخال Enterللانتقال للسطر التالى وهذا ما يحدث فى كثير من الامثلة مثل حالة تنفيذ الامر dir/p فى نظام التشغيل DOS حيث يتم عرض الفهرس صفحة بصفحة وينتظر الضغط على أى مفتاح للاستمرار وعند الضغط لا يظهر الحرف المضغوط عليه ولكن ينفذ الامر فقط.

مثال :

```
char a;
a=getch();
printf("%c",a);
```

دالة طباعة حرف واحد putchar ()

ملف التوجيه : stdio.h

تستخدم لطباعة حرف واحد على الشاشة فمثلا الصورة: putchar('a'); تطبع على الشاشة الحرف a كما هو

والمثال الموجود فى (شكل ٧-٢) يوضح كيفية استخدام دوال إدخال حرف وكذلك دالة طباعة حرف حيث يستقبل حرف باستخدام كل دالة لتوضيح الفرق بينهم.

```
0:    /* Program Name : cs2_7.c */
1-1:  #include <stdio.h>
1-2:  #include <conio.h>
2:    main ( )
3:      {
4:          char ch1,ch2,ch3;
5:          printf("\n ch1=");
```

```
6:        ch1=getchar();
7:        printf("\n ch2=");
8:        ch2=getche();
9:        printf("\nch3=");
10 :      ch3=getch();
11:       putch(ch1);
12:       putch(ch2);
13:       putch(ch3);
14:    }
```

شكل ٧-٢ استخدام دوال إدخال وطباعة حرف

وفى هذا البرنامج

- فى السطر رقم ٥ الدالة ( ) printf تطبع =ch1 بينما يشتمل السطر رقم ٦ على الدالة ()getchar وهى تستقبل حرف وتخزنه فى المتغير ch1
- وبالمثل نفهم السطور ٧و٨ والسطور ٩و١٠ مع ملاحظة الفرق فى طبيعة كل دالة
- ويشتمل السطر رقم ١٠ على الدالة ()getch التى تسمح باستقبال حرف من لوحة المفاتيح الا أن هذا الحرف لايظهر على الشاشة.
- وفى السطور رقم ١١و١٢و١٣ الدالة ()putch وتستخدم لطباعة الحروف المخزنة بالمتغيرات ch3,ch2,ch1

عند تنفيذ البرنامج ستحصل على النتيجة التالية

```
ch1=a
ch2=b
ch3=
a
b
c
```

دالة gets() والدالة puts()

الخطو التالية في اتجاه زيادة قوة الدوال نتناول من خلالها دالتي gets() ، puts() وهي تسمح بكتابة وقراءة تعبيرات نصية strings في نظام console .

الدالة gets() تقرأ جملة نص string المدخلة من لوحة المفاتيح وتضعها في الذاكرة عند الموضع المشار إليه بمؤشر تأخذه الدالة كوسيطة مع ملاحظة أن الدالة لا تتوقف سوى بعد ضغط مفتاح Enter ... وما قبل ضغط Enter تستطيع استخدام مفتاح Back space لمسح الحرف الأخير.

وعلى سبيل المثال البرنامج التالي يقرأ نص ثم يطبع طوله باستخدام الدالة strlen( ) في مكتبات لغة C التي تحسب طول تعبير نصي.

```
#include <stdio.h>
#include <string.h>
main(void)
{
char str [80]; // إعلان عن مصفوفة حروف سعتها 80
gets (str);
printf("\nlenght is %d\n", strlen(str));
return 0
}
```

الدالة puts()

هذه الدالة تكتب التعبير النص المعطى لها على الشاشة متبوعاً بسطر خالي وشكلها العام بمعنى أنها لا تحتاج إلى \n في نهايتها لكي تنهي السطر تلقائياً فإن كتبت بيانات أخرى تأتي من بداية السطر التالي.

وهي تتعرف على جميع الرموز المستخدمة مع (\) كما في دالة printf() فمثلاً

\t تعني **tab**

\n تعني سطر جديد ... الخ

ويحتاج استدعاء puts بجهود أقل من حيث المعالجة procesing عن دالة printf() ذلك لأن puts() تكون مخرجاتها فقط أحرف وليس في استطاعتها إخراج أرقام أو التحويل باستخدام محددات الشكل format specifiers مثل %d ، %c ... الخ.

كما يحدث مع printf() ولذلك فهي أسرع في التنفيذ وتحتاج حيز أقل في الذاكرة وتعيد الدالة مؤشر للنص المعطى لها بعد وضعه على الشاشة فمثلاً الجملة التالية تظهر كلمة Welcome على الشاشة.

```
Puts ("Welcome");
```

والجدول التالي يلخص الدوال السابقة واستخداماتها

الدالة	استخدامها
getchar()	تقرأ حرف من لوحة المفاتيح – تنتظر ضغط لـ
getche()	تقرأ حرف مع إظهاره على الشاشة – لا تنتظر ضغط لـ
getch()	تقرأ حرف دون إظهاره على الشاشة – لا تنتظر ضغط لـ
putchar()	تكتب حرف على الشاشة
gets()	تقرأ نص من لوحة المفاتيح وتستمر في تلقي النص حتى ضغط لـ
puts()	تكتب نص على الشاشة

دوال Console I/O المنسقة

بالإضافة إلى مجموعة الدوال السابقة والتي تعرف على أنها دوال بسيطة لا تسمح بالتنسيق format في المخرجات توجد دالتين تسمحان بتنسيق الوسيطات وهي دالة printf() ، دالة scanf().

وهما تعملان بنفس الطريقة غير أن printf() تكتب بيانات إلى Console بينما scanf() تقرأ بيانات وهي تتعامل مع جميع أنواع البيانات سواء حروف أو أرقام.

دالة printf() قد سبق الحديث عنها من قبل حتى تستطيع التطبيق عليها داخل الفصول السابقة.

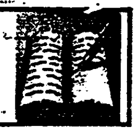
دالة scanf()

هي الدالة المقابلة لدالة printf() ، وتعد هي الدالة الرئيسية في نظام إدخال البيانات فيما يخص console حيث أنها تشمل الصورة العامة التي تعد أي دالة إدخال بيانات حالة خاصة منها.

وهي تتعامل كما في printf() مع

1 - محددات الشكل format specifier .

2 - المسافات الخالية white space characters .



محددات الشكل هي مجموعة رموز تستخدم بعد رمز % وتخبر دالة scanf() عن نوع البيانات الذي سيقرأ بعدها فعلى سبيل المثال %d تقرأ بيانات integer في حين %s تجهز الدالة لقراءة string .

محددات الشكل فقد علمنا ما تعنيه وكيف تستخدم أما بالنسبة للمسافات الخالية في دالة scanf() تستخدمها الدالة في تخطي واحد أو أكثر من الرموز في قناة الإدخال input stream .

وتتعامل scnf() مع المسافات الخالية سواء كانت (مسافة ، tab ، سطر خالي) نفس المعاملة وينتج عنها كما ذكرنا تخطي كود حرف أو أكثر (على حسب عددها) في قناة الإدخال input stream .

وتأخذ الدالة نوعين من الوسيطات الأولى هو محددات نوع البيانات مثل %d للاستعداد لأخذ رقم أو %S للاستعداد لأخذ تعبير نص أو حتى %C لأخذ حرف واحد ، وبعدها تأخذ عناوين الذاكرة للمتغيرات التي ستوضع فيها المدخلات.

والجدير بالذكر أن جميع المتغيرات التي تتلقى قيمها من خلال scanf() يجب أن تكون مؤشرات لأن scanf() تعيد عنوان الذاكرة فإن أردت للمتغيرات أن تحمل قيمة يجب أن تستخدم مؤشر & في إدخال القيمة من scanf() فمثلاً لحفظ رقم صحيح في المتغير count يجب أن تكتب الجملة على الصورة.

```
Scanf("%d" , &count);
```

بالنسبة للنصوص ... فإنها تقرأ في مصفوفة حروف character array واسم المصفوفة بدون أي رقم بين قوسيها يعني عنوان عنصر المصفوفة الأول ، لذلك لكي تقرأ نص في مصفوفة اسمها item تكتب الجملة على الصورة

```
scanf("%s" , item);
```



في هذه الحالة اسم المصفوفة item هو في حد ذاته مؤشر للمصفوفة لذلك لا مجال لاستخدام المؤثر &.

يجب فصل البيانات المدخلة بواسطة مسافات ، tab أو أسطر خالية لأن علامات (; ,) لا يمكن أخذها كفواصل بين المدخلات بمعنى أن الجملة.

```
Sacnf(%d%d" , &m , &n);
```

تقبل القيمة 1050 لكنها لن تسمح بإدخال 10.50 فكما كان يحدث مع printf() تكون أشكال المحددات مرتبطة بنوعية المتغيرات التي سيأتي بعدها في قائمة الوسيطات.

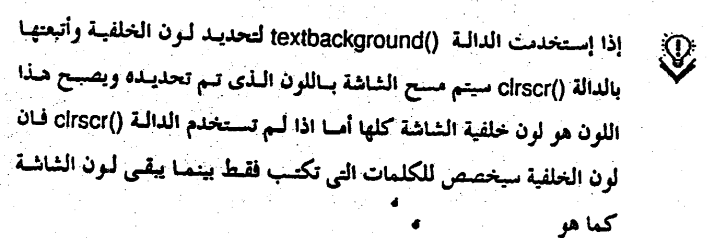
بذلك نكون قد تعرضنا لمعظم الدوال المستخدمة من خلال console وهي بالطبع ليست جميع الدوال لكن أهمها وأكثرها استخداماً.

دوال الإدخال والإخراج التي تستخدم الألوان

سبق أن أشرنا إلى أن دالة الطابعة printf() ودالة الإدخال scanf() وكذلك باقي الدوال التى تم شرحها قد صممت بحيث تعمل باللون الابيض على الاسود ولا تتأثر بدوال تغيير الالوان فإن هناك مجموعة من الدوال المقابلة للدوال السابقة والتى صممت للتعامل بالالوان المحددة وكلها مسبوقة بالحرف c مثل

cprintf(),cscanf(),cputs(),cgets(),.....................

وكلها معرفة في الملف conio.h



إذا إستخدمت الدالة ()textbackground لتحديد لون الخلفية وأتبعتها بالدالة ()clrscr سيتم مسح الشاشة باللون الذى تم تحديده ويصبح هذا اللون هو لون خلفية الشاشة كلها أما اذا لم تستخدم الدالة ()clrscr فان لون الخلفية سيخصص للكلمات التى تكتب فقط بينما يبقى لون الشاشة كما هو .

نفذ البرنامج الموجود بشكل ١١-٢ ولاحظ الألوان التى ستحصل عليها

```
 0:  /*Program Name : cs2_11 */
 1:  #include <stdio.h>
 2:  #include <conio.h>
 3:  main ()
 4:     {
 5:        textbackground(BLUE);
 6:        clrscr();
 7:        textcolor(RED);
  :       cprintf("\r\n  THIS TEXT  DISPLAYED WITH RED ON
                                              BLUE COLOR");
 9:       getch();
10:      textbackground(GREEN);

11:      clrscr();
12:      textcolor(YELLOW);
  :      cprintf("\r\n  THIS TEXT  DISPLAYED WITH YELLOW
                                            ON RED COLOR");
14:  }
```

شكل ١١-٢ برنامج لتغيير الألوان

وعن هذا البرنامج نوضح مايلى

- فى السطر رقم ٥ تغير الدالة ;(textbackground(BLUE لون خلفية الكتابة الى اللون الازرق
- فى السطر رقم ٦ تمسح الدالة ;()clrscr الشاشة (يصبح لونها أزرق)
- فى السطر رقم ٧ الدالة ;(textcolor(RED تغير لون الكتابة الى اللون الاحمر وهكذا

لاحظ \r\n : تقابل \n فى حالة الدالة ()printf

دوال التعامل مع الطابعة

من الأمور المهمة للمبرمج المبتدىء والمحترف على السواء إخراج البيانات على الطابعة والتعامل مع الطابعة.

ولغة C تمكننا من التعامل مع الطابعة من حيث إخراج البيانات وإختبار حالة الطابعة وهل هى تعمل أم مغلقة وهل الورق موجود بالطابعة أم لا وفى الفقرة التالية سوف نتعرض لأبسط صورة للتعامل مع الطابعة وهى الطباعة فقط أما باقى الحالات فسنتعرض لها فيما بعد

وتستخدم الدالة ()fprintf للطباعة على الطابعة ولكن مع معرف الطابعة القياسى stdprn ويظهر كأول معامل فى الدالة ()fprintf والمتغير stdprn يعنى الطابعة ومعناه اطبع هذه الرسالة على الطابعة ومعنى ذلك أن هناك معرفات أخرى لها معنى مختلف سوف نتعرض لها لاحقا.

وتشبه الدالة ()fprintf الدالة ()printf حيث تستخدم نفس المعاملات التى تستخدمها الا أن المخرجات تظهر على الطابعة. وكذلك باقى صور الدالة ()printf مسموح بها مع ()fprintf

جرب البرنامج التالى :

```
0:  /* Program Name :cs2_12.c
1:  #include <stdio.h>
2:  main ()
3:    {
4:      fprintf(stdprn," These words go to the printer............");
5:    }
```

شكل ١٢-٢ برنامج للطباعة على الطابعة

ستحصل على الرسالة التالية :

These words go to the printer على الطابعة.

تذكر

عندما تريد	إستخدم الدالة
طباعة حروف أو أرقام	printf()
طباعة عبارات حرفية فقط	puts()
إدخال حروف أو أرقام	scanf()
إدخال عبارات حرفية	gets()
إدخال حرف دون إظهاره على الشاشة	getch()
إدخال حرف مع إظهاره على الشاشة	getche()
مسح الشاشة	clrscr()
تغيير موضع المؤشر	gotoxy()
تغيير ألوان الكتابة	textcolor()
تغيير ألوان خلفية الكتابة	textbackground()
الطباعة بالألوان على الشاشة	cprinf(),cputs
إدخال البيانات بالألوان	cscanf(),cgets
الطباعة على الطابعة	fprintf()


الفصل الرابع
مفهوم الدوارات
فى لغة C++

الفصل الرابع

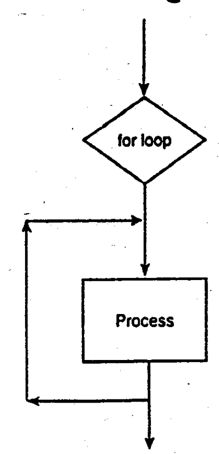
مفهوم الدورات Loops في لغة C++

مفهوم loop، وأهم استخداماتها:

تتطلب العديد من العمليات الحسابية وجود مجموعة من التعليمات المتتالية التى تتكرر تنفيذها إلى حين تحقيق شرط معين يكون المستخدم قد حدده مسبقاً. وتعرف هذه العملية باسم iteration (التكرار)، ولكنها تأخذ اسماً آخر فى لغة C++، وهو: Loop. وهناك ثلاثة أنواع منها: for loop وwhile loop وdo while loop.

والآن نعرض للـloop الأولى منها (For loop): تتخذ For loop عند تنفيذها المسار الذى يوضحه الشكل القادم حيث يتم الدخول إلى الصندوق العلوى المعروف باسم For loop، ويتم فيه تحديد عدد المرات المطلوب تكرار الأمر فيها. ثم ينتقل بعد ذلك إلى الصندوق السفلى

المعروف باسم process، ويتكرر تنفيذ كافة التعليمات الموجودة فيه. ويشير عدد مرات التحرك من الصندوق الأول إلى الثانى إلى عدد مرات تكرار تنفيذ الأمر نفسه. ويستمر ذلك إلى حين تحقيق شرط معين. وفى النهاية يتم الخروج من هذه الحلقة على النحو الموضح فى الشكل التالى:



المسار الذى تتخذه for loop عند تنفيذها

كيفية تغيير قيم المتغيرات فى البرنامج:

فى جميع لغات البرمجة، من الممكن تعديل قيم المتغيرات على مدار تشغيل البرنامج، وذلك بأحد الأسلوبين التاليين: الرفع و(incrementing) أو الخفض (decrementing). فلو كان هناك- مثلاً- متغير اسمه x من نوع integer، فيمكن رفع قيمته بلغة ++C بالجمل البرمجية التالية:

```
x = 10 ;
x = x + 1 ;
```

وبذلك تتغير قيمة المتغير x لتصبح 11 بدلاً من 10، ويكون بذلك قد تم رفع قيمة المتغير بمقدار واحد صحيح. والصيغة المختصرة لتنفيذ ذلك هى:

```
x ++ ;
```

وبالمثل، يمكن خفض قيمة المتغير بمقدار واحد صحيح بكتابة الجملة البرمجية التالية:

```
x = x - 1 ;
```

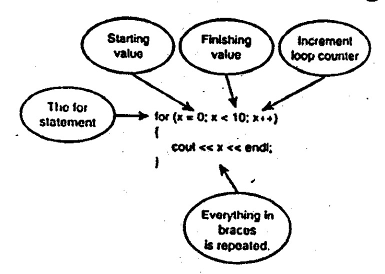
أو بالصيغة المختصرة لهذا الأمر وهى:

```
x -- ;
```

وبذلك لو كانت قيمة المتغير الأولى 10، ستصبح 9.

تركيب جملة for Loop:

تبدأ جملة for loop بتحديد الإسم المحدد للمتغير، ثم توضع القيمة المبدئية له وهى الصفر، ثم يوضع الشرط الذى يحدد عدد المرات التى سيتم فيها رفع قيمة المتغير من الصفر وحتى أقل من عشرة. ويكون ذلك بالجملة التالية: ; ++ x. ويوضح الشكل التالى ذلك حيث يتكرر تنفيذ جميع الأوامر التى اشتملت عليها الحاصرتين } { عشر مرات (0 : 9).



قواعد تركيب جملة for loop

وفى المثال التالى الترجمة العملية لما جاء فى الفقرة السابقة . حاول تشغيل البرنامج ، وستظهر أمامك على الشاشة الأعداد من 0 إلى 9 فى شكل عمود أسفل يسار الشاشة .

مثال (٥-١): بيان للكيفية التى يتم بها تنفيذ جملة for loop من النوع البسيط.

```
#include <iostream.h>

main()
{
    int x;

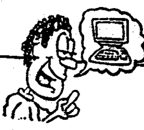
for (x = 0; x < 10; x++)
{
cout << x  << endl;
}
return(0);
```



- حاول ذلك -

تذكر أن عدد المرات التى تم تنفيذ loop فيها يكون محكوماً بقيم المتغير x فى بداية ونهاية البرنامج. وعليك أن تستخدم القيم التالية بدلاً من القيم المذكورة فى المثال السابق، ولاحظ النتائج:

```
for (x = 0 ; x < 15 ; x ++)
for (x = 0 ; x < 10 ; x ++)
for (x = 0 ; x < 1 ; x ++)
```



تحذير:

لا توضع علامة الفاصلة المنقوطة (;) فى نهاية سطر جملة for loop على الإطلاق. فوضعها على هذا النحو الخاطئ:

```
for (x = 0 ; x < 10 ; x ++) ;
```

معناه أن البرنامج ستتم له عملية compilation، وعند تشغيله لن يتم تنفيذ الخطوات المراد تكرار التعليمات فيها. وسيحذف البرنامج جميع الجمل الموجودة بين الحاصرتين } {. والسبب فى ذلك أن وجود الفاصلة المنقوطة يعنى فى لغة الكمبيوتر أن جملة loop انتهت. ويعتبر ذلك من الأخطاء الشائعة فى تعلم لغة C++.

كيفية تنفيذ برنامج Running Total مع المزيد من القيم المختصرة:

سنعرض الآن لعملية أخرى فى لغة ++C تعرف باسم finding a running total. وهى عبارة عن إيجاد مجموع جاه من القيم الداخلة إلى البرنامج. وسيعرض مثال (٥-٢) لكيفية تنفيذ هذه العملية على خمس متغيرات من نوع integer.

مثال (٥-٢): نموذج لبرنامج Running Total، الذى يتم الحصول فيه على مجموع جاد من قيم المتغيرات الداخلة خلال Runtime:

```
#include <iostream.h>

main()
{
    int loop;
    int total = 0;      // declare and initialize variables
    int number = 0;

    for (loop = 1; loop <= 5; loop++)
    {
      cout << "Enter a number : ";
      cin >> number;
      total += number;          // keep running total
    }
    cout << endl << "The total is " << total;
    return(0);
}
```

وتتلخص خطوات البرنامج فى وجود متغيرين : number لحفظ القيمة الداخلة وtotal لحفظ الناتج. ثم تحدد القيمة المبدئية للمتغير (0) فى بداية البرنامج. وهى نفسها العملية التى سبق وأشرنا إليها باسم initialization of variables (تكوين المتغيرات).

وسيطلب من المستخدم فى بداية loop أن يدخل قيمة معينة باستخدام أمر cin. أما المتغير total، فسيكون بمثابة الصندوق الذى يحتفظ بمجموع هذه الأعداد الداخلة للمتغير.

وستكون القيمة المبدئية للتغير فى المرة الأولى من تنفيذ loop صفراً، ويتم بعدها ضافة القيمة الداخله إلى المتغير number إلى القيمة الداخله إلى المتغير total ثم يوضع لناتج فى المتغير total . وتكرر العملية نفسها عند تنفيذ loop للمرة الثانية . وبما أن المتغير total أصبح الآن يحتوى على ناتج المجموع السابق، فسيكون من الممكن إضافته إلى القيمة الجديدة للمتغير number وإيجاد مجموع جار جديد، وتتكرر العملية خمس مرات إلى حين إيجاد القيمة الإجمالية لمجموع الخمس أعداد معًا .



المقصود بكلمة Bugs فى لغة الكمبيوتر

تعنى كلمة Bugs فى اللغة الإنجليزية الحشرة أو الذبابة، ولكنها تستخدم فى لغة الكمبيوتر بمعنى الخطأ الذى قد يحدث فى أى جزء من أجزاء البرنامج. واشتق منها مصطلح debug ومعناه: تصحيح الأخطاء فى كتابة البرنامج.

وإليك الآن مجموعة من الصيغ المختصرة المستخدمة فى برامج إيجاد المجموع الجارى فى لغة C++ :

```
total + = number;
```

وهو اختصار الجملة البرمجية التالية :

```
total = total +  number;
```

ويتعنى هذه الجملة البرمجية أن محتويات المتغير total قد تم إضافتها إلى محتويات المتغير number ، ووضع الناتج بعد ذلك فى هذا المتغير الأول total . ويلاحظ أن كلا التركيبين يؤديان العمل نفسه فى البرنامج .

ومن الأفضل دائمًا البدء بأعداد بسيطة مثل: 1، 2، 3 . وذلك للتأكد من سلامة كتابة البرنامج قبل الدخول فى أعداد مركبة .



المزيد من عمليات الحساب

ويمكن إجراء العمليات الحسابية المختلفة على قيم المتغيرين السابقين، وبنفس الكيفية السابق ذكرها.

١- لطرح قيمة المتغير number من قيمة المتغير total، ووضع الناتج فى المتغير total، تكتب الجملة التالية:

```
total -= number;
```

٢- لضرب قيمة المتغير number فى قيمة المتغير total، ووضع الناتج فى المتغير total ثانية، تكتب الجملة التالية:

```
total *= number;
```

٣- لقسمة قيمة المتغير number على قيمة المتغير total، ووضع الناتج فى المتغير total، تكتب الجملة التالية:

```
total /= number;
```

كيفية تحويل نص البرنامج من حالة Uppercase إلى حالة Lowercase والعكس:

سيتم العرض فى الفصل الرابع عشر "معالجة أنواع مختلطة من البيانات باستخدام Structure" لتكنولوجيا معينة خاصة بقاعدة البيانات. وثمة مشكلة كبيرة ستواجه المستخدم، وسأشير الآن للحل المقترح لها. من المعروف فى لغة الكمبيوتر أنها تتعامل مع المدخلات على أنها افتراضات. فلو تم - مثلاً - إدخال اسم كوكب ما بأشكال مختلفة من التراكيب:

VENUS
Venus
venus
VcNuS

سيتعامل الكمبيوتر معها على أنها تشير إلى أربع كواكب مختلفة . وليس إلى كوكب احد .

ويمكن لحل هذه المشكلة أن يحدد المستخدم منذ البداية شكلاً معينًا لكتابة هذه لأسماء وتخزينها فى ذاكرة الجهاز . وبما أن الاسمين الثانى والرابع ليسا من الأساليب صحيحة فى البرمجة ، فلا يبقى أمامنا إلا الاسمين الأول والثالث ، والذين يمثلان كلين من أشكال كتابة نص البرنامج هما : أسلوب Lowercase وأسلوب Uppercase الكتابة بالحروف الاستهلالية الكبيرة والصغيرة) . ولا يصح أبدأ الخلط بينهما فى البرنامج واحد ، ولابد من تحديد أسلوب واحد منهما فقط عند كتابة البرنامج لأول مرة .



أهمية تحديد شكل الكتابة فى نص البرنامج:

لعله اتضح من المثال السابق كيف يتعامل الكمبيوتر مع النصوص المكتوبة بأشكال مختلفة، الأمر الذى يؤكد للمستخدم عدم الخلط بينهما فى نص واحد. والسبب الوحيد فى ذلك أن المستخدم سيحتاج حينئذ للعديد من سطور الكود ليتحقق من صحة البيانات التى تم إدخالها من قاعدة البيانات. وبما أن ذلك قد يؤدى إلى إهدار الكثير من الأموال، وحدوث أكثر من خطأ فى البرنامج الواحد، فإن الحل الأسلم لذلك: تحديد شكل واحد لكتابة جميع الحروف منذ البداية.

وعلى ذلك ، فإن أى اسم يدخله المستخدم إلى البرنامج سيكتب بالشكل السابق حديده لنمط الكتابة فى هذا البرنامج . ويمكن استخدام جمل for loop لاختيار صحة لشكل الذى كتب به كل حرف على حده . ويعرض مثال (٥-٣) لعدد من الأساليب ... التى سيتم العرض لها بعد قراءة البرنامج .

مثال (٥-٢): كيفية تحويل البيانات الداخلة من حالة Uppercase إلى حالة Lowercase :

```
#include <iostream.h>
#include <string.h>
```

```
#include <ctype.h>
#define MAX 20

main()
{
   char name[MAX];
   int x;

   cout << "Enter a name : ";
   cin >> name;
   for (x = 0; x < strlen(name); x++)
   {
      name[x] = toupper(name[x]);
   }
   cout << endl << name << endl;
   return(0);
}
```

ويلاحظ فى هذا المثال النقاط التالية :

(أ) يستخدم البرنامج نوعين جديدين من libraries هما : string.h وctype. يتعامل النوع الأول مع بيانات من نوع string فقط ، بينما يتعامل النوع الثانى مع أنواع مختلفة من البيانات يتم معالجتها بلغة ++C .

(ب) توجد فى البرنامج الجملة البرمجية التالية :

```
char name [MAX];
```

ولكن سيتم العرض لوظيفة هذه الجملة فيما بعد.

(ج) يستخدم البرنامج أمر strlen (name)، وهو أحد الأوامر الموجودة فى string.h library ، ووظيفته : تحديد المدى الخاص بهذا المتغير name . وتغيير القيمة بعد ذلك لاستكمال جملة for loop .

(ء) يستخدم البرنامج أيضاً جملة ;(name [X] =toupper (name [X] ، ويتم فيها تغيير شكل الكتابة فى المتغير X من Lowercase (حروف صغيرة) إلى Uppercase (حروف استهلالية كبيرة) .

والآن، ادخل البرنامج، واختبره بإدخال مجموعات مختلفة من الحروف الا ستهلالية الكبيرة والصغيرة، وسيكون الناتج دائمًا حرفًا استهلاليًا كبيراً لأن ذلك ما حددته منذ البداية . ويمكنك أيضاً إدخال بعض الأعداد، وملاحظة النتائج .

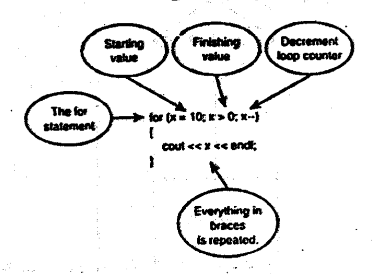


كيفية التحويل إلى Lowercase:

ويمكنك استبدال أمر toupper السابق ذكره فى البرنامج بأمر tolower، وذلك من أجل تحويل نص البرنامج من uppercase إلى lowercase .

إستخدام جملة for loop فى خفض قيم المتغيرات:

لم تعرض حتى الآن إلا لشكل واحد من تنفيذ for loop، وهو رفع قيمة المتغير : ١، ٢، ٣، ٤، وهكذا. ولكن، هناك استخدام آخر لها، وهو خفض قيمة المتغير : ٤، ٣، ٢، ١، وهكذا. ويأخذ ذلك التركيب التالى :



قواعد تنفيذ جملة for loop بنظام العد التنازلى لخفض قيم المتغيرات

ويلاحظ أنها هي نفسها القواعد السابق العرض لها في تركيب جملة for loop الأصلية. ولكن، باستثناء شيء واحد فقط: أن المتغير قد دخل بقيمة أكبر من القيمة التي انتهى إليها، أي أن المستخدم قد قام هنا بخفض قيمة المتغير بدلاً من رفعها.

ويمكنك تنفيذ البرنامج التالي لملاحظة النتائج:

مثال (٥-٤): العد التنازلي في جملة for loop.

```
#include <iostream.h>

main()
{
    int x;

    for (x = 10; x > 0; x--)
    {
     cout << x  << endl;
    }
     return(0);
 }
```

أهم الدروس المستفادة:

- تستخدم جملة for loop لتنفيذ جميع عمليات التكرار المطلوبة في البرنامج.
- Iteration: مصطلح عام يشير إلى عملية تكرار تنفيذ الأمر أكثر من مرة في البرنامج، وتعتبر for loop تطبيقًا على ذلك.
- تأخذ جملة for loop في لغة C++ التركيب التالي:

```
for (x=0;x<6;x++)
```

- تستخدم الجملة التالية لرفع قيمة متغير ما في البرنامج:

```
x=x+1
```

والصيغة المختصرة لها هي x++.

الدوارات المتداخلة باستخدام for

الدورات المتداخلة عبارة عن دوارة كبيرة تشتمل بداخلها على دوارة أو أكثر. بمعنى أن مجموعة التعليمات الموجودة بالدوارة الداخلية يتكرر تنفيذها طالما لم ينته العداد فاذا انتهى ينتقل التنفيذ إلى الدوارة الخارجية ويتكرر تنفيذ تعليمات الدوارة الخارجية حتى ينتهى العداد المحدد لها . وتشبه فكرة الدورات المتداخلة فكرة عمل عقارب الساعه فتجد عقرب الثوانى يدور ٦٠ دورة فيدور عقرب الدقائق بمقدار دقيقه وهكذا.

والبرنامج الموجود فى الشكل رقم ٨-٣ يستخدم فكرة الدورات المتداخلة لطباعة الأرقام من صفر إلى ٩ باستخدام الدوارة الداخلية ثم يطبع الأرقام من صفر إلى ٥ باستخدام الدوارة الخارجية. بحيث نحصل فى النهاية على طباعة رقم من الدوارة الخارجية مقابل طباعة عشرة أرقام من الدوارة الداخلية.

```
0:/* Program Name : cs3_8.c  */
1: #include <stdio.h>
2:  main()
3:  {
4:    int i,j;
5:     for(i=0;i<5;i++)
6:       {

7:       for(j=0;j<10;j++)
8:          {
9:              printf("\n i=%d",i);
10:               printf("\t j=%d",j);
11:          }        /*  first for loop     */
12:       }           /*  secound for loop */
13:  }                /*  main() function   */
```

شكل ٨-٣ استخدام الدوارات المتداخلة

ومن الامثلة المفيدة لاستخدام الدورات المتداخلة استخدام الدوارة for لطباعة جدول الضرب من أول ١×١ الى ١٢ × ١٢ حيث نستعمل الدوارة الخارجية لتغيير

الرقم المضروب ونستعمل الدوارة الداخلية لتغير الرقم المضروب فيه من ١ الى ١٢ كما يتضح من البرنامج الموجود فى الشكل رقم ٩-٣

```
0:/* Program Name :cs3_9.c  */
1: #include <stdio.h>
2:  main()
3:  {
4:     int i,j;
5:      for(i=1;i<13;i++)
6:        {
7:          printf("\n");
7:          for(j=i;j<13;j++)
8:             {
9:                 printf(" IxJ=%d",i*j);
10:
11:            }     /*  first for         */
12:        }         /*  second for        */
13:  }               /*  main() function   */
```

شكل ٩-٣ برنامج استخدام الدوارات المتداخلة لطباعة جدول الضرب

أكتب البرنامج ونفذه وتأكد من النتيجة

ولتحسين البرنامج بحيث لا يطبع القيم المكررة مثل ١×٢ و ٢×١ نقوم بتغير بداية الدوارة الثانية بحيث تبدأمن القيمة i. فيصبح السطر رقم ٧ كما يلي:

```
for (j = i;j <13;j++)
```

الدوارة اللانهائية بإستخدام While

ومعناها تكرار تنفيذ الجملة بدون شرط ولايتوقف التنفيذ حتى يضغط المستخدم Ctrl+c او Ctrl+break وتأخذ الدوارة اللانهائية باستخدام for الصورة

```
for(;;)
```

والمثال التالى يستعمل الدوارة اللانهائية للاستمرار فى طباعة كلمة Allah مالم يضغط المستخدم على Ctrl+break فاذا ضغط المستخدم على Ctr+c تنتهى الدوارة ويتوقف التنفيذ

```
#include <stdio.h>
main ()
  {
   for(;;)
    printf("\t Allah");
  }
```



لو كتبت الدوارة بالصورة (;;)for وأتبعتها بالعلامة ; وحاولت تنفيذ الملف النهائى (EXE.) فان الجهاز يدخل فى مسار لانهائى(hang) ولاتستطيع حل هذه المشكلة الا بغلق الجهاز وتشغيله مرة أخرى ويمكنك استخدام هذه الفكرة مع برنامج كلمة السر فإذا أدخل المستخدم كلمة سر خطأ لا يستجيب الجهاز لاى أمر ولابد من غلقه.

الدوارة while

تستخدم الدوارة while لتكرار تنفيذ جملة أو مجموعة جمل عدد من المرات غير معلوم العدد وانما يتوقف هذا العدد على شرط موجود بالدوارة وتأخذ الدوارة while الصورة التالية :

```
while(condition)
   statment;
```

ومعناها طالما أن الشرط (condition) صحيح نفذ الجملة (Statement)

وهى تقوم بتكرار الجملة أو مجموعة الجمل التابعة لها طالما كان شرط التكرار صحيح وعندما يصبح شرط التكرار غير صحيح يتوقف تنفيذ الدوارة

والبرنامج الموجود فى الشكل ١٠-٣ يستخدم الدوارة while لطباعة كلمة Allah ١٠ مرات كل مرة فى سطر مستقل وهذا الاستخدام للدوارة while غير جيد ولكننا نكتب هذا البرنامج لتوضيح أجزاء الدوارة while

```
0: /* Program Name : cs3_10.c  */
1:#include <stdio.h>
2:  main ()
3:    {
4:      int  i=0;
5:      while(i<10)
6:        {
7:           printf("\n Aallah");
8:           i++;
9:        }
10:   }
```

شكل ١٠-٣ استخدام while

وعن هذا البرنامج

- في السطر رقم ٤ إعلان عن متغير من نوع صحيح هو i
- السطر رقم ٥ معناه طالما أن قيمة i أقل من ١٠ نفذ الجمل التالية وبالفعل أول مرة ستكون قيمة لمتغير i تساوى صفر وبالتالي يتم تكرار الدوارة
- السطر رقم ٧ معناه نفذ الجملة ;("printf("\n Allah طالما أن الشرط صحيح وهو طباعة كلمة Allah
- السطر رقم ٨ زيادة قيمة المتغير i بمقدار ١ ثم يعود البرنامج الى اختبار شرط الدوارة while فاذا كانت قيمة i ماتزال أقل من ١٠ يتكرر تنفيذ الجمل التالية وهكذا حتى يصبح الشرط غير صحيح

ملاحظات

١. لاحظ أن كتبنا القوس } بعد while لاننا نرغب فى تكرار التنفيذ على أكثر من جملة (بلوك) والقوس { فى سطر مستقل لاغلاق الدوارة
٢. لتغير قيمة الزيادة أو جعلها بالسالب غير التعبير ++i كما نشاء كما تم مع الدورة for

الفرق بين for و while

الدوارة for دوارة عددية حيث تعتمد على العداد وينتهى التكرار فيها بانتهاء عدد مرات التكرار أما الدوارة while فدوارة شرطية أى تعتمد على الشرط الذى يلى

الأمر while حيث تتكرر الجمل التي تليها طالما كان الشرط صحيحا وتنتهي الدوارة بكسر هذا الشرط. وبالتالي الاستخدام الامثل للدوارة for هو تكرار عملية أكثر من مرة بشرط أن يكون عدد مرات التكرار معلوم والاستعمال الامثل للدوارة while هو التكرار بناء على شرط معين

مثال

يقوم البرنامج الموجود بشكل ١١-٣ باستقبال حروف من لوحة المفاتيح ويستمر في ذلك حتى تضغط مفتاح Enter وعند ضغط مفتاح Enter يتوقف قبول الحروف ويطبع البرنامج عدد الحروف التي أدخلتها.

```
1/* Program Name : cc3_11.c */
2:  #include <stdio.h>
3:  #include <conio.h>
4:  main ()
5:    {
6:       int count=0;
7:       printf ("\n ENTER CHARCTERS..\n");
8:       while (getche() !='\r')
9:           count++;
10:      printf ("\n charcter count is %d ",count);
11:   }
```

شكل ١١-٣ برنامج عد الحروف المدخلة

وعن البرنامج الموجود بالشكل ١١-٣ نوضح مايلي :

في السطر رقم ٨ تقوم الدالة getche() باستقبال حرف وتقوم الدوارة while باختبار هذا الحرف فاذا كان هذا الحرف هومفتاح الادخال Enter يتوقف عمل الدوارة واذا كان أي حرف آخر يقوم السطررقم ٩ بزيادة قيمة المتغير count وهكذا حتى يضغط المستخدم على مفتاح الادخال وفي النهاية يطبع البرنامج عدد الحروف المدخلة.

الدوارة اللانهائية باستخدام while

سبق أن قلنا أن الدوارة اللانهائية مع for تأخذ الصورة (;;) for أما الدوارة اللانهائية مع while فتأخذ الصورة (while(1 وهي أكثر استعمالا من الدوارة for والبرنامج الموجود بشكل ١٢-٣ يطبع قيم متوالية ٠و١و٢ وهكذا وذلك باستعمال دوارة لانهائية حتى يضغط المستخدم على Ctrl+C.

```
/* Program Name : cs3_12.c  */
   #include <stdio.h>
    main ()
      {
         int i=0;
         while(1)
            {
              printf ("i=%d",i);
              i++;
```

شكل ١٢-٣ الدورة اللانهائية

الدوارة Do .... While

تستخدم الدوارة do.... while لتكرار تنفيذ جملة أو مجموعة جمل أكثر من مرة بناء على شرط معين كما هو الحال مع الدوارة while ولكن الفرق بينهما أن الدوارة while تختبر الشرط أولا فاذا كان صحيحا تنفذ الجمل التالية لها والا فلا ، أما الدوارة do.....while فتنفذ الجمل التالية لها أولا ثم تختبر الشرط. فاذا كان صحيحا تعيد التنفيذ والا توقف التكرار.

وتأخذ الدوارة do ..... while الصيغة

```
do
 {
   statement1;
 }while (condition);
```

ومعناها do أى نفـذ الجمـل التاليـه وهى Statement1 ومـا يليهـا طالمـا كـان الشـرط (Cndertion) صحيح.

والبرنامج الموجود بشكل ١٣-٣ يستخدام الدوارة do ...while لطباعة رسالة يطلب فيها من المستخدم ادخال كلمة سر ومقارنتها بالكلمة المخزنة فاذا كانت الكلمـة المدخلة صحيحة انتهى البرنامج والا يتم تكرار استقبال كلمة السر مرة اخرى.

```
1:  /* Program name : cs3_13.c  */
2:  /* password...*/
3:  #include <stdio.h>
4:  #include <conio.h>
5:     main ()
6:       {
7:          char pass[10];
8:          do
9:           {
10:            printf ("\n Enter Password:");
11:            scanf ("%s",pass);
12:           } while (strcmp(pass, "azab") !=0);
13:       }
```

شكل ١٣-٣ برنامج كلمة السر

وعن هذا البرنامج نوضح مايلي:

- في السطر رقم ٨ بداية الدوارة do والسطر رقم ١٠ يطبع الرسالة Enter Password: على الشاشة
- والسطر رقم ١١ يستقبل كلمة ويخزنها في المتغير pass والسطر رقم ١٣ يقارن بين الكلمة التي أدخلت وكلمة azab فاذا كانتا متطابقتين ينتهي عمل do..while والا عاد البرنامج الى بداية do.
- نفذ البرنامج وأدخل كلمة سر غير الواردة بالبرنامج وهي azab مرة والثين قبل كتابة كلمة السر الصحيحة ولاحظ تنفيذ البرنامج.



الدالة () strcmp تقوم بمقارنه متغيرين من نوع عبارة حرفيه string فإذا كان المتغيرين متطابقين كان الفرق بينهما صفر.

مثال

البرنامج الموجود بالشكل ١٤-٣ تعديل لبرنامج كلمة السر السابق بحيث لا تظهر كلمة السر التي يكتبها المستخدم وهي تكتب على الشاشة حتى لايراها شخص آخر وتعتمد فكرته على تغير الالوان.

اكتب البرنامج ونفذه ولاحظ ذلك.

```
1:  /* Program Name : cs3_14.c  */
2:  /* password...*/
3:    #include <stdio.h>
4:    #include <conio.h>
5:    main ()
6:       {
7:           char ch;
8:           char pass[10];
```

```
9:     do
10:      {
11:       textcolor(WHITE);
12:       textbackground(BLUE);
13:       cprintf ("\r\n Enter Password:");
14:       textbackground(WHITE);
15:       cscanf ("%s",pass);
16:      } while ((strcmp(pass,"azab")) !=0);
17:   }
```

شكل ١٤-٣ برنامج كلمة سر لا تظهر على الشاشة

وعن هذا البرنامج نوضح مايلى :

يقوم البرنامج في السطر رقم ١١و١٢ بتحديد لون الكتابة أبيض ولون الخلفية أزرق

- والسطر رقم ١٣ يطبع العبارة Enter Password بهذه الالوان.
- السطر رقم ١٤ يجعل لون الخلفية أبيض وبالتالى يتم الكتابة بالون الابيض على خلفية بيضاء فلا تظهر كلمة السر وهذا هو المطلوب.
- ثم يعاود البرنامج تغير الألوان وهكذا حتى يدخل المستخدم كلمة السر الصحيحة.

مثال (٦-٤): كيفية كتابة الأعداد المحصورة بين 0 إلى 9 باستخدام جملة do while loop:

```
#include <iostream.h>

main()
{
     int counter = 0;              //set initial value

     do
     {
          cout << counter << " ";      //display
          counter++;                 //increment
     }while ( counter < 10);            //test condition
     return(0);
}
```

أهم الدروس المستفادة:

- while loop هى الطريقة الثانية من طرق التكرار فى لغة ++C . ويتم فيها تكرار تنفيذ جميع الأوامر الموجودة فى جملةloop طالما أن الشرط لايزال صحيحًا . وأحيانًا تنتهى loop دون تنفيذ أية أوامر .
- تكتب جملة while loop فى لغة ++C بالتركيب التالى :

```
while (counter ++ <10)
```

- لتجنب اللجوء إلى الغلق المفاجئ للجهاز والذى يضيع معه كل ما تم تنفيذه من عمل سابق فى البرنامج ، لابد من التأكد من أن الشرط الذى سيتم اختياره قد تم تعديله فى الجزء المؤثر من loop .
- فى حالة استخدام رمز السابقة يتم رفع قيمة المتغير أولاً ، ثم يتم التحقق من صحة الشرط بعد ذلك .
- فى حالة استخدام رمز اللاحقة يتم التحقق من الشرط أولاً ، ثم ترفع قيمة المتغير بعد ذلك .
- do while loop هى الطريقة الثالثة من طرق التكرار فى لغة ++C . ويتم فيها تنفيذ جميع الأوامر الموجودة فى جملة loop طالما أن الشرط لا يزال صحيحًا . وعادة ما يتم تنفيذ هذه الأوامر مرة واحدة فقط على الأقل .

تذكر

عندما تريد	إتبع الآتى
عمل دوارة تعتمد على عداد معلوم بدايته ونهايته	إستخدم الدوارة for
عمل دوارة تعتمد على شرط غير معلوم	إستخدم الدوارة while
تنفيذ الدوارة أولا ثم الشرط	إستخدم الدوارة do.....while
تنفيذ أكثر من عبارة داخل البلوك	ضع القوس } فى بداية البلوك و القوس { فى نهاية الدوارة


الفصل الخامس
الجمل الشرطية
والتفريع

الفصل الخامس
الجمل الشرطية والتفريغ

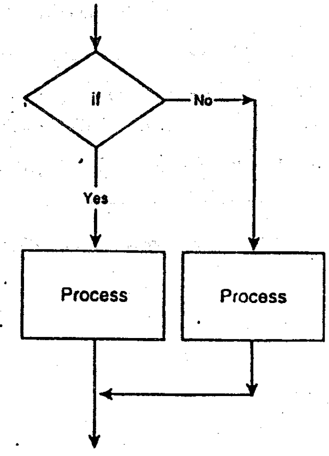
كيفية الاختيار بين المسارات فى لغة ++C:

فى هذا الفصل، سنعرض لجملة أخرى من الجمل البرمجية الموجودة فى لغة ++C، وهى جملة if else . وسيتضح بالضبط ما هو المقصود بهذه الجملة البرمجية، والكيفية التى تستخدم بها فى البرنامج المكتوب بلغة ++C. كما سيوضح الشكل القادم أيضاً تركيب الجملة، والمسار الذى تتخذه عند تنفيذها.

مسار جملة if else:

تستخدم جملة if else فى لغة ++C لإتاحة الاختيار بين مسارات تنفيذ إحدى العمليات. وهى تقنية من شأنها أن تزيد من جودة وكفاءة البرنامج المكتوب بهذه اللغة.

وكما هو موضح فى الشكل، فإن مسار هذه الجملة يبدأ بعمل نوع من المقارنة يتم من خلالة تقرير صحة الشرط من خطأه، وهى العملية التى سبق وعرضنا لها فى الفصل السادس "آلية تنفيذ جملة while loop" : فلو تحقق الشرط، ستسير العملية فى المسار "Yes" لتنفيذ الأوامر الخاصة به. ولو لم يتحقق الشرط فيثبت خطأه، تسير العملية فى المسار "No" لتنفيذ الأوامر الخاصة به وحده. وخلاصة القول أنه إن تم اختيار المسار "Yes"، فإن ذلك يعنى تنفيذ جميع الأوامر فى جزء if. أما اختيار المسار "No"، فيعنى تنفيذ الأوامر فى جزء else .

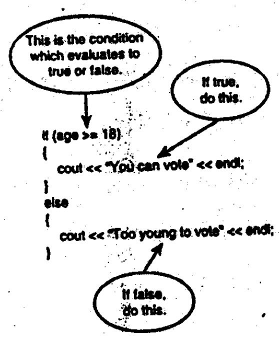


المسار الذى تتخذه جملة if else

تركيب جملة if else:

يعرض الشكل التالى للكيفية التى يتم بها كتابة جملة if else فى لغة C++. وهى كيفية تؤكد على مدلول أداء هذه الجملة فى البرنامج . فإن تحقق الشرط، يتم تنفيذ جميع أوامر المجموعة الأولى من الحاصرات ويتم التغاضى عن الأوامر التالية لذلك فى جزء else.

وبالمثل، لو لم يتحقق الشرط، يتم تنفيذ جميع أوامر المجموعة الثانية من الحاصرات، والتجاوز عن الأوامر فى جزء if السابق له . وعلى ذلك، يمكن القول بأن جملة if else لا تتيح إلا تنفيذ عملية واحدة فقط؛ الأمر الذى يؤدى إلى رفع مستوى جودة وكفاءة الأداء.



تركيب جملة if else

مثال تطبيقى على جملة if else:

يقدم مثال (٧-١) عينة لبرنامج يتم فيه إدخال بيانات خاصة بعمر أحد الأشخاص. والمطلوب: إجراء مقارنة من المقارنات الموجودة فى لغة C++ لتحديد ما إذا كان يصح له الإدلاء بصوته فى الانتخابات أم لا . فإن كان عمره من ثمانية عشر عامًا فما أكثر، يصح له التصويب. وإن كان عمره أقل من الثمانية عشر عامًا، لا يصح له التصويت. حاول تنفيذ البرنامج، ولاحظ أى المسارين سيتم اختياره لتنفيذ العملية.

مثال (٧-١): برنامج بسيط يستخدم جملة if else:

```
#include <iostream.h>

main()
{
    int age;

    cout << "Enter your age : ";
    cin >> age;

    if (age >= 18)
```

```
    {
        cout << "You can vote " << endl;
    }
    else
    {
        cout << "Too young to vote" << endl;
    }
    return(0);
}
```

أنواع الجمل البرمجية فى لغة ++C:

استخدم البرنامج السابق نوعاً واحداً فقط من الجمل البرمجية ، وهو النوع البسيط الذى يتكون من سطر واحد من الكود يتم من خلاله تنفيذ أمر واحد فى البرنامج وقد تمثل هذا النوع فى الإختيارين البديلين اللذين أتاحتهما جملة if else لتنفيذ البرنامج السابق الخاص بشروط التصويت فى الإنتخابات . وهاتين الجملتين هما :

```
cout <<"you can vote "<<endl;
cout <<"Too young to vote "<<endl;
```

ولكن الأمر لا يسلم من وجود أكثر من أمر واحد فى البرنامج يكون الكود الخاص به مكتوباً فى عدد من السطور . وهذا ما يطلق عليه مفهوم الجملة البرمجية المركبة . وهى عبارة عن جملة برمجية واحدة مكونة من كود متعدد السطور يؤدى أمراً واحداً فى البرنامج ، وعادة ما يوضع الكود الخاص به فى مجموعة واحدة من الحاصرات لتيسير قراءتها وفهمها على المستخدم . ومثال ذلك :

```
{
        cout <<"Hello I am your local senator" << endl;
        cout << "You are over 18" << endl;
        cout << "That means you can vote" << endl;
        cout << "I will crawl to you" << endl;
}
```



متى تحذف الحاصرات؟

من الممكن جداً حذف الحاصرات فى الجمل البرمجية البسيطة. وذلك لأنه ما من وظيفة تؤديها فى البرنامج إلا تيسير قراءة الكود.

والآن، انظر للمثال التالى، ولاحظ كيف تم استخدام الجمل البرمجية المركبة التى تعمل كاختيارين بدلين لتنفيذ جملة if else.

مثال (٢-٧): نموذج لكيفية استخدام الجمل البرمجية المركبة فى جملة if else.

```
#include <iostream.h>

main()
{
    int age;

    cout << "Enter your age : ";
    cin >> age;

    if (age >= 18)
    {
        cout << "Hello I am your local senator" << endl;
        cout << "You are over 18" << endl;
        cout << "That means you can vote" << endl;
        cout << "I will crawl to you" << endl;
    }
    else
    {
        cout << "Too young to vote" << endl;
        cout << "Do not waste my time" << endl;
    }
    return(0);
}
```

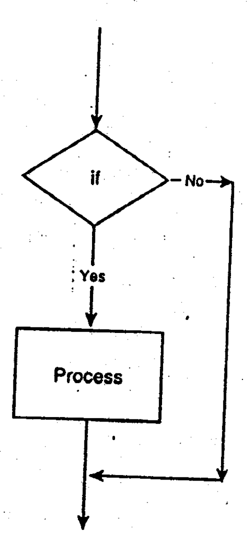


ماذا لو حدث خطأ فى وضع الحاصرات؟

لو لم توضع الحاصرات فى المكان الصحيح من الكود، سيحدث خطأ عند تشغيل البرنامج، ولا سيما مع الجمل البرمجية المركبة. لذا، فلابد للمستخدم من توخى الحذر عند وضع هذه الحاصرات فى البرنامج.

مزيد من الإيضاح لجزء else :

عرفنا فى الأمثلة السابقة أنه فى حالة عدم تحقق الشرط يتم الإنتقال مباشرة إلى الأوامر الموجودة فى جزء else لتنفيذها . ولكن ، ماذا لو لم توجد أية أوامر بديلة فى جملة else ؟ عندئذ لا يكون هناك أوامر على الإطلاق ، ومن ثم ، يتم الخروج من جملة if else كلها إلى الجزء الذى يليها فى نص البرنامج . ويوضح الشكل التالى ذلك :



مسار تنفيذ جملة if else فى حالة عدم وجود أوامر فى جزء else

يعرض المثال التالى لنموذج برنامج لا يشتمل على جزء else

مثال (٧-٣) : برنامج لا يشمل على أى نوع من الأوامر البديلة التى يتم تنفيذها فى حالة عدم تحقق الشرط :

```
#include <iostream.h>

main()
{
    int age;

    cout <<"Enter your age : ";
    cin >> age;
```

```
    if (age >= 18)
    {
        cout << "Hello I am your local Senator" << endl;
        cout << "You are over 18" << endl;
        cout << "That means you can vote" << endl;
        cout << "I will crawl to you" << endl;
    }
    return(0);
}
```

مفهوم Nesting فى لغة ++C:

لعله قد اتضح من الأمثلة السابقة مدى السهولة التى تتيحها جملة if else لعمل اختيارات بديلة لأداء البرنامج. ولكن فى كثير من الأحيان يضطر المبرمجون إلى عمل أكثر من اختيار بديل واحد لتنفيذ عمليات البرنامج. ومن هنا دعت الضرورة إلى ايجاد وسيلة أخرى تتيح الاختيار بين هذه الأداءات المختلفة. ونطلق على هذه التقنية اسم nesting. وقد تم تعديل جملة if else العادية بهذه التقنية ليصبح اسمها nested if else بدلاً من normal if else.



مامعنى "Nested"؟:

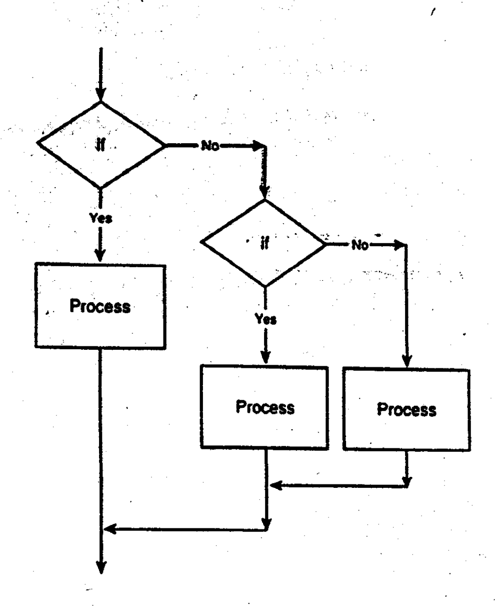
المقصود بمفهوم Nesting أو nested if else أن هناك جملة أخرى من if else تم وضعها داخل جملة if else الأولى. وبمعنى آخر: فقد أصبحت هذه الجملة الثانية متداخلة مع الأولى، وهذا هو التعريف ذاته الذى تجده فى المعجم لكلمة nesting. ومن الممكن استخدام هذه التقنية فى معالجة Loop.

مسار تنفيذ جملة nested if else:

بالنظر إلى الشكل التالى، يتضح أن مسار تنفيذ جملة nested if else يتلخص فى نقطتين اثنتين:

- فى حالة تحقق الشرط: يتم تنفيذ جميع الأوامر الموجودة فى المسار "yes" تماماً مثل جملة if else العادية.

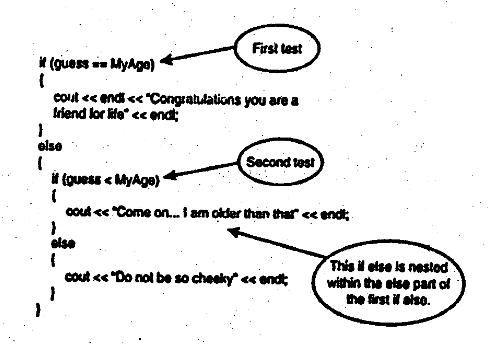
• فى حالة مالم يتحقق الشرط : يتم اتخاذ المسار "(No)" لتنفيذ العملية الموجودة فى جملة if else الداخلية . وبذلك يكون قد تم الدخول فى جملة nested if else ، وتتكرر العملية بالكيفية نفسها التى سبق وذكرناها فى جملة if else العادية .



جملة if else المركبة.

تركيب جملة nested if else:

الكود التالى هو ترجمة عملية لما تم العرض له فى الشكل السابق . فهو تعرض للكيفية التى تكتب بها جملة nested if else فى البرنامج ، والتى تكون بالشكل التالى :



تركيب جملة nested if else.

مثال تطبيقى على تنفيذ جملة nested if else:

بفرض أنك قد كتبت جميع البيانات العمرية بالشكل الصحيح، يمكنك الآن الدخول وتشغيل البرنامج التالى الذى سيتم فيه التحقق من صحة البيانات العمرية الداخلة إليه، ثم عرض رسالة معينة على الشاشة بذلك.

مثال (٤-٧): نموذج لبرنامج يستخدم جملة nested if else:

```
#include <iostream.h>

#define MyAge 21    // Don't laugh!

main()
{
    int guess;

    cout <<"Guess my age : ";
    cin >> guess;
    if (guess == MyAge)
    {
            cout << endl
                << "Congratulations you are a friend for life"
                << endl;
    }
    else
    {
     if (guess < MyAge)
```

```
{
        cout << "Come on...I am older than that" << endl;
    }
    else
    {
        cout << "Do not be so cheeky" << endl;
    }
  }
  return(0);
}
```

وتتلخص خطوات البرنامج فى النقاط التالية :

• كتابة جملة الشرط الذى سيتم التحقق منه فى شكل المقارنة التالية :

```
if (guess == MyAge)
```

• لو أثبتت المقارنة صحة الشرط ، ستظهر رسالة معينة على الشاشة تأتى نتيجة لتنفيذ الأوامر الموجودة فى جزء if الخاص بجملة if الأولى .

• لو أثبتت المقارنة عدم صحة الشرط (سواء لزيادة البيانات العمرية الداخلة عن العمر المطلوب أو لنقصانها) ، سيتم الدخول فى جملة if else الأخرى ويختبر الشرط الموضوع فيها بالمقارنة التالية :

```
if (guess < MyAge)
```

• لو أثبتت المقارنة صحة الشرط ، يتم تنفيذ الأوامر الخاصة بهذا الاختيار البديل .

• لو ثبتت المقارنة عدم تحقق الشرط الثانى أيضاً ، يتم اللجوء إلى الاختيار الوحيد الباقى ، وهو أن البيانات العمرية الداخلة أقل من العمر المطلوب ، وستظهر رسالة معنية على الشاشة توضح ذلك .

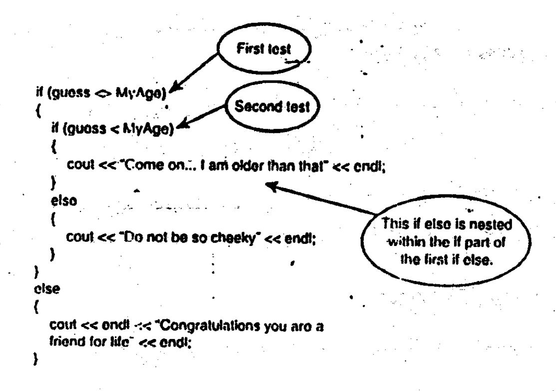
تركيب ثان من تراكيب جملة nested if else:

جملة nested if else ليست قاصرة على الجزء الثانى فقط من جملة if else الخارجية ، فقد توجد أيضاً فى الجزء الأول منها . ويوضح الشكل التالى ذلك :



مفهوم program fragment:

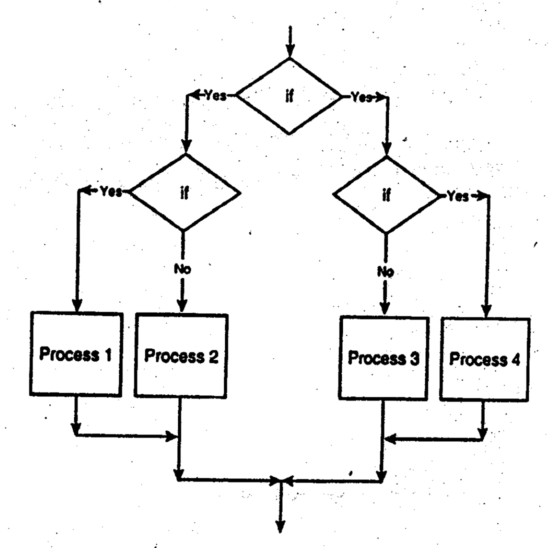
يقصد بهذا المفهوم الجزء الصغير من الكود الذى رغم عدم كفايته للعمل بمفرده فى البرنامج إلا أنه يستخدم أحياناً لتوضيح إحدى النقاط الهامة الخاصة به دون الإضطرار لحذف الباقى منه. وعادة ما يكون ذلك هو الجزء الذى يحتوى على( ) main .



تركيب جملة nested if else عندما تأتى فى الجزء الأول من جملة if else الخارجية.

تركيب ثالث من تراكيب جملة nested if else:

عرضنا فى التركيبين السابقين لموضعين من مواضع التداخل بين جملتى normal if else و nested if else . والآن نعرض للتركيب الثالث . فقد تأتى جملة nested if else فى كلا جزئى جملة if else العادية . وتعرف الجملة فى هذه الحالة باسم double compound if بينما كانت الأولى تعرف باسم compound if فقط . ويكون مسار تنفيذ الجملة فى هذه الحالة بالشكل التالى :



مسار تنفيذ جملة nested if else فى كلا جزئى جملة if else العادية.

حاول تنفيذ البرنامج التالى الذى يتم فيه إدخال بيانات عمرية خاصة ببعض الأشخاص. وستلاحظ بعد تشغيله، ظهور رسالة من الرسائل التالية حسب الفئة العمرية التابع لها كل شخص منهم:

الفئة العمرية	*الرسالة*
أقل من ٥	تحت سن المدرسة
أقل من ١٦	فى سن المدرسة
أقل من ٦٥	فى سن العمل
من ٦٥ فما أكثر	فى سن التفرغ

والآن، ادخل وشغل هذا البرنامج لترى كيف تستخدم جمل if else العادية لحل هذه المشكلة. ولاحظ أن الفكرة فى هذا البرنامج اختبار مرحلة العمر المتوسط بكتابة هذا الشرط:

if else <16

وبذلك، يكون قد تم حصر كافة الاختيارات فى جزئين اثنين، يتشعب كل منهما إلى جزئين آخرين حتى يتم الوصول إلى حل لهذه المسألة :

مثال (٥-٧): نموذج لبرنامج يستخدم جملة double compound if:

```
#include <iostream.h>

main()
{
   int age;

   cout << "Enter your age : ";
   cin >> age;

   if (age < 16)
   {
     if ( age < 5)
     {
       cout << "Under school age" << endl;
     }
     else
     {
       cout << "School age" << endl;
     }
   }
   else
   {
     if (age < 65)
     {
       cout << "Working age" << endl;
     }
     else
     {
       cout << "Over the hill" << endl;
     }
   }
   return(0);
}
```

التفريغ غير المشروط

التفريغ غير المشروط معناه الانتقال إلى مكان محدد داخل البرنامج بدون شرط وتقوم جملة goto بهذا الغرض وتأخذ الشكل العام التالى :

Goto lb :

حيث lb متغير من نوع char يشير إلى المكان المطلوب الانتقال اليه ولاننصح باستخدام هذه الجمله لأنها تستخدم بكثرة مع اللغات الغير تركيبية مثل لغة البيسك أما في حالة اللغة التركيبية (لغة C) فيفضل ستخدام الدوال لتغير مسار تنفيذ البرنامج.

مثال :

البرنامج الموجود بالشكل ١٠-٤ يستخدم جملة goto للتفريغ غير المشروط داخل البرنامج وفيه نلاحظ أن البرنامج سيطبع كلمة Allah باستمرار لان التفريغ غير مشروط أى لا يوجد شرط لانهاء تكرار التنفيذ فاذا رغبت فى انهاء البرنامج فيجب الضغط على Ctrl+C

```
0:  /*Program Name : cs4_10.c  */
1:  #include <stdio.h>
2:  main ()
3:     {
4:       char  RE;
5:  RE:
6:       printf("\t allah");
7:       goto RE;
8:     }
```

شكل ١٠-٤ التفريغ الغير مشروط باستخدام أمر goto

التفريع switch ....case

لاحظت فى المثال السابق أن إستخدام جملة if فى حالة تعدد الإختيارات لأكثر من إختيارين يمثل عبئا على المبرمج فى تتبع خطوات البرنامج ويسبب بطئا نسبيا فى تنفيذ البرنامج لذا إستخدمنا الجملة الشرطية if...else if ويمكن استعمال التفريغ switch ... case كبديل لجملة if ... else if وهى طريقة أسهل كما سنرى وتستخدم بالصيغة التالية

```
ch=getch();
switch (ch)
{
 case '1':
   statement1;
   statement2;
   ....
   break:
 case '2':
   statement1;
   statement2:

   ....
   break;
default:
   statement1;
   statement2;
   ...
 }
```

فى هذا التركيب تختبر جملة switch المتغير OP ثم تفرض له مجموعة حالات هذه الحالات تحدد بكلمة case ، ففى الحالة الاولى (: "case "a) إذا كانت قيمة المتغير OP هى a يتم تنفيذ الجمل التالية والموجودة فى هذه الحالة ثم الخروج من تركيب .. switch بإستخدام كلمة break ، بالمثل الحالة الثانية وهكذا. أما إذا لم تحقق حالة من الحالات يتجه التنفيذ إلى كلمة default وهى بمعنى إذا لم يتحقق أى حالة من الحالات السابقة نفذ مايلى:

وننصح باستعمال هذا الاسلوب فى حالة اختيار حالة من مجموعة حالات .

والبرنامج الموجود بالشكل ٨-٤ تعديل لبرنامج الالة الحاسبة السابق والموجود بالشكل ٧-٤ ولكن باستخدام التركيب switch ... case

```
0:/* Program Name : cs1_8.c */
1:  #include <stdio.h>
2:  #include <conio.h>
3:  main ()
4:   {
5:      float num1,num2;
6:      char ch,op;
7:      do
8:       {
9:        clrscr();
10:       printf ("\n Type num1 op num2:");
11:       scanf ("%f %c %f",&num1,&op,&num2);
```

٧٩

```
12:       switch (op)
13:        {
14:         case '+':
15:            printf ("sum=%f",num1+num2);
16:            break;
17:         case '-':
18:            printf ("sub=%f",num1-num2);
19:            break;
20:         case '*':
21:            printf ("mul=%f",num1*num2);
22:            break;
23:         case '/':
24:            printf ("div=%f",num1/num2);
25:            break;
26:         default:
27:            printf ("\n unknowen operator..");
28:        }
29:     printf ("\n again (y/n):");
30:    }while ((ch=getch()) =='y');
31:  }
```

شكل ٨-٤ استخدام التركيب switch...case

وعن هذا البرنامج نوضح مايلى :

- فى السطر رقم ١١ الدالة scanf() تستقبل ثلاث متغيرات رقمين ، ومؤثر (علامة حسابية) ، و فى السطر ١٢ الجملة Switch تختبر المتغير opوفى السطر رقم ١٤ كلمة case تحدد حالة ما اذا كان المؤثر (العلامة الحسابية) هو علامة الجمع + فاذا كان المؤثر هو علامة الجمع + ينفذ السطر رقم ١٥ وهو جمع الرقمين وطباعة النتيجة ثم الخروج من الاختيارات عن طريق break وهكذا السطور حتى السطر رقم ٢٨

- فى السطر رقم ٣٠ دالة getch() تستقبل حرف وتختبره جملة while اذا كان y يعود التنفيذ مرة أخرى من أول السطر رقم ٩ والا انتهى تنفيذ البرنامج
- ومن هذا البرنامج نجد أن التركيب case ... switch أكثر وضوحا من التركيب if..else if.

وعند تنفيذ البرنامج نحصل على النتيجة التالية

```
Type num1 op num2:55+55
sum=110
again (y/n):y
Type num1 op num2:100-50
sub=50
again (y/n):n
```

مثال

من التطبيقات المشهورة لاستخدام التفريع switch()....case هو استخدامه فى قوائم الاختيارات (menu) كما فى الشكل التالى :

```
MAIN MENU
        1-INTRODUCTION TO C-LANG.
        2-STRUCTUTE OF C-PROG.
        3-LOOPS
        4-DECISIONS.
        ENTER SELECTION:
```

ويحتوى شكل ٩-٤ على البرنامج المطلوب لإظهار هذه القائمة والتعامل معها

```
0:  /* Program Name : cs4_9.c */
1:  #include <stdio.h>
1:  #include <conio.h>
2:  main()
3:     {
4:      int  se;
6:       do {
7:         clrscr();
8:         printf("\n     MAIN MENU");
9:         printf("\n            1-INTRODUCTION TO C-LANG.");
10:        printf("\n            2-STRUCTUTE OF C-PROG.");
11:        printf("\n            3-LOOPS");
12:        printf("\n            4-DECISIONS.");
13:        printf("\n           ENTER SELECTION:");
14:        scanf("%d",&se);
15         switch(se)
16:             {
17:               case 1:
18:                      printf("\n 1-INTRODUCTION TO C-LANG.");
19:                      break;
20:               case 2:
21:                      printf("\n   2-STRUCTUTE OF C-PROG.");
22:                      break;
23:               case 3:
24:                      printf("\n  3-LOOPS");
25:                      break;
26:              case 4:
24:                      printf("\n   4-DECISIONS");
25:                      break;
26:             default:
27:                      printf("\n\n unknowen selection");
28:                      break;
29:             }
30:         printf("\n\n again(y/n)==>");
31:      } while(getch()=='y');
32:  }
```

شكل ٤-٩ برنامج باستخدام التركيب switch...case

وعن هذا البرنامج نوضح مايلى:

- السطور من ٨ الى ١٣ تطبع القائمة الرئيسية
- السطر رقم ١٤ يستقبل رقم صحيح ويخزنه فى المتغير se
- فى السطر رقم ١٥ يختبر المتغير se ويحدد قيمته
- السطر ١٧ يختبر قيمة المتغير se فإذا كانت الرقم ١
- السطر ١٨ يطبع الرسالة التالية 1-INTRODUCTION TO C-LANG
- وهكذا حتى السطر رقم ٢٩ وهو آخر سطر فى تركيب Switclh ... Case
- السطر ٣٠ يطبع الرسالة again(y/n)==> والسطر ٣١ يستقبل حرف ويختبره فإذا كان حرف y تعيد الدوارة while تنفيذ البرنامج مرة أخرى وهكذا

المؤثر الشرطى ? (Conditional operator)

هذا المؤثر يقوم مقام جملة if

ويأخذ الشكل الاتى :

```
var=(condition) ? num1:num2;
```

ومعناه

```
if condition is true  then var = num1
if condition is false then var = num2
```

أى اذا كان الشرط صحيح فإن var=num1 واذا كان الشرط غير صحيح فإن var=num2

وهو بديل لتركيب if الذى يأخذ الشكل الآتى :

```
if (condition)
        var=num1;
  else
        var=num2;
```

مفهوم menu-driven program (برنامج يعمل بنظام القوائم):

فى العديد من البرامج يتم عرض قائمة بالخيارات المطلوبة، التى يختار منها المستخدم ما يشاء. ويقوم الكمبيوتر بعد ذلك بتنفيذ الأمر الذى وقع عليه اختيار المستخدم. وقد عرضنا جمل if else و switch case التى تتيح عمل هذه الاختبارات، ولعلك لاحظت مدى السهولة التى تتيحها جملة switch case ، والتى لا تقتصر على سهولة الكتابة فقط، بل وسهولة الفهم أيضاً إذا ما أراد المستخدم قراءة الكود مرة أخرى.

وسيتضح من المثال التالى الكيفية التى يتم بها تنفيذ هذا النوع من البرامج. وهو الأمر الذى سأوجز له فى مجموعة من النقاط قبل عرض البرنامج. وكل ما يجب أن تعرفه الآن أن هذا البرنامج الذى سيعرض له المثال القادم يعرف فى لغة الكمبيوتر باسم front end (الواجهة الأمامية)، وهو خاص بنظام الحجز فى الخطوط الجوية.



ما المقصود بعبارتى Front End و Back End؟

front end: كل ما يراه المستخدم الأخير للبرنامج على الشاشة، ويتفاعل معه. و back end: كل العمليات الخاصة ببرمجة الجهاز، والتى من شأنها تنفيذ كافة الوظائف المطلوبة فيه. وهو جزء لا يتسنى للمستخدم الشعور به أو رؤيته، وهو ما يمكن الاصطلاح عليه باسم آلية العمل فى الجهاز.

آلية تنفيذ برنامج يعمل بنظام القوائم:

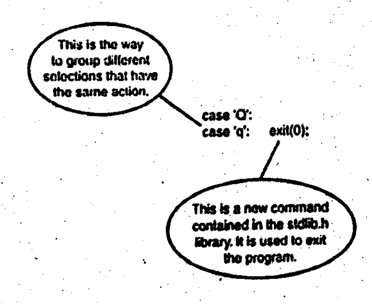
يقدم مثال (٨-٣) نموذجاً لبرنامج يعمل بنظام القوائم، وتتلخص خطوات تنفيذه فيما يلى:

- تقديم نص مكتوب عبارة عن عدد من السطور التى تمثل الاختيارات المتاحة للمستخدم فى البرنامج.
- يحدد المستخدم الأمر الذى يريد تنفيذه.
- تختبر جملة switch case هذا الأمر بمقارنته مع قائمة الاختيارات المتاحة فى البرنامج.
- يدخل هذا الاختبار فى جملة do while loop، وتتكرر المقارنة بين الأمر وخيارات القائمة حتى يتم نقر خيار exit للخروج من البرنامج. وهو أمر موجود فى stdlib.h library.

والآن، حاول تشغيل هذا البرنامج مع أخذ كافة النقاط السابق ذكرها فى الاعتبار.

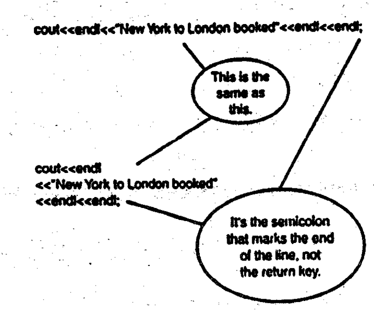
بعض التقنيات الخاصة التى استخدمها البرنامج السابق:

يلاحظ فى البرنامج السابق أنه قد اشتمل على أسلوبين من أساليب تيسير قراءة الكود فى لغة ++C. والأسلوب الأول أنه فى حالة ما إذا كان هناك خيارين يشيران لأمر واحد، فإنه يتم كتابة كلا الخيارين واحداً تلو الآخر. ومثال ذلك: أمر quit الذى عبر عنه الخيار Q والخيار q:



خيار أمر quit:

وبالنسبة للأسلوب الثانى الذى ييسر أمر قراءة الكود، فيتمثل فى إمكانية أن يتصدر سطر الكود مجموعة من السطور دون حدوث أى خلل فى تشغيل البرنامج ويوضح الشكل التالى ذلك:





الفرق بين الخيار Q والخيار q:

الفارق الوحيد بين الخيار Q والخيار q يكون فى استخدام مفتاح caps lock الموجود بلوحة المفاتيح. وهو مفتاح يستخدم للتمييز بينهما، لأنه لو ترك الأمر للمستخدم، سيؤدى ذلك للكثير فى إمكانية.

مثال (٨-٢): نموذج لبرنامج يعمل بنظام القوائم:

```
#include <iostream.h>
#include <stdlib.h>

main()
{
   char choice;

   do
     {

    // This is the menu displayed to the screen

    cout << "FLIGHT BOOKING SYSTEM" << endl << endl;
    cout << "1..New York to London Heathrow" << endl;
    cout << "2..New York to Vancouver" << endl;
    cout << "3..New York to Sydney" << endl;
    cout << "4..New York to Cape Town" << endl;
    cout << "Q..Quit" << endl;

    // This reads in the user selection.

    cout << endl << endl << "Enter your choice : ";
    cin >> choice;

    // This section acts on the user selection.

    switch (choice)
    {
       case '1' :    cout << endl
                       << "New York to London booked"
                       << endl << endl;
                     break;

       case '2' :    cout << endl
                       << "New York to Vancouver booked"
                       << endl << endl;
                     break;

       case '3' :    cout << endl
                       << "New York to Sydney booked"
                       << endl << endl;
                     break;

       case '4' :    cout << endl
                       << "New York to Cape Town booked"
                       << endl << endl;
                     break;

       case 'Q' :
       case 'q' :    exit(0);
     }
   } while(1);
   return(0);
}
```

رسالة الكمبيوتر بعد تنفيذ البرنامج:

وستظهر أمامك على الشاشة القائمة التالية:

```
FLIGHT BOOKING SYSTEM

1..New York to London Heathrow
2..New York to Vancouver
3..New York to Sydney
4..New York to Cape Town
Q..Quit

Enter your choice :
```

وعلى ذلك، فلو نقرت الخيار الرابع - مثلاً، ستظهر هذه الرسالة أمامك على الشاشة:

```
New York to Cape Town booked
```

نقطة أخيرة:

ما هو نوع البيانات التى أدخلها المستخدم فى اختياره؟ بالطبع هى بيانات من نوع integer، ولكن لاحظ أن القيم قد كتبت فى القائمة هكذا: '1'، '2'، '3'، '4'. ولفهم أنها من نوع char، فلابد من قراءتها هكذا: 1، 2، 3، 4.



متى تحول مدخلات int إلى char؟

عندما تكون البيانات الداخلة إلى البرنامج من نوع char، فإن كل ما يتم إدخاله من لوحة المفاتيح يكون اسمه ASCII. ويعنى ذلك باختصار American Standard Code for Information Interchange (الكود القياسى الإمريكى لتبادل المعلومات). ويتراوح مدى الكود من هذا النوع بين 0 و255.

ويمثل ذلك عدد مرات الضغط على لوحة المفاتيح. فعند تحديد متغير ما بأنه من نوع char، يتم إدخال قيمة له من خلال لوحة المفاتيح، ويقوم البرنامج المكتوب بلغة C++ بعد ذلك بتحويل المدخلات من نوع ASCII إلى نوع char. ويحدث الشئ نفسه مع المدخلات من نوع integer وfloat ولكن دون شعور المستخدم بذلك.

أهم الدورس المستفادة:

- تستخدم جملة switch case لعمل قائمة بكافة الشروط المطلوبة لأحد متغيرات البرنامج. ويتم تنفيذ الأمر إذا تحقق تماثل بين ما تم إدخاله وإحدى الشروط فى القائمة.
- تتعقد جمل nested if else بكثرة الإستخدام، بينما تظل جمل switch case قصيرة وواضحة.
- الشرط الوحيد لاستخدام جملة switch case أن تكون القيم الموضوعة فى القائمة من نوع البيانات الدالة على الترتيب الأبجدى أو العددى.
- تعد جملة switch case من أفضل الجمل المستخدمة لكتابة البرامج التى تعمل بنظام القوائم.
- يمكن فى حالة وجود أكثر من خيار للأمر الواحد أن تكتب هذه الخيارات واحداً بعد الآخر، بينما تكتب الأمر مرة واحدة.
- أمر exit أحد الأوامر الموجودة فى stdlib.h library، ويستخدم للخروج من البرنامج.
- ASCII اختصار لمصطلح American Standard Code for Information Interchange (الكود القياسى الأمريكى لتبادل المعلومات). وتشير بيانات char من هذا النوع إلى مدى رقمى معين للضغط على لوحة المفاتيح يتراوح من عدد 0 إلى 255 ضغطة.



الفصل السادس
المصفوفات

الفصل السادس
المصفوفات

معنى المصفوفات

تنقسم البيانات إلى بيانات حرفية (char) وبيانات رقمية (int) وبيانات حقيقيه (float) وتسمى هذه الأنواع (int,float,char) بالأنواع الرئيسيه للبيانات ، حيث لايمكن تجزئتها أقل من ذلك.

ولكن هناك أنواع أخرى من البيانات تسمى بالأنواع المشتقة (Drived data types) من هذه الأنواع المصفوفات Arrays. تعرف المصفوفة بأنها مجموعه من العناصر تنتمى إلى نوع واحد. ويخصص لها اسم واحد وتنقسم المصفوفات الى مصفوفات ذات بعد واحد ومصفوفات ذات بعدين

والمصفوفة ذات البعد الواحد مثل :

A=[3 4 5 7 9]

وتسمى مصفوفة ذات بعد واحد لأنها تتكون من صف واحد أو عمود واحد، وفيها حرف A هو اسم المصفوفة ،والارقام هى عناصر المصفوفة ويتم الاشارة الى كل عنصر برقم العنصر أى بترتيبه داخل المصفوفة على أن يبدأ العد بالرقم صفر كما يلى العنصر A[0] يساوى ٣ و العنصر A[1] يساوى٤ والعنصر A[2] يساوى ٥ وهكذا ..

والمصفوفة ذات البعدين تأخذ الشكل التالى :

$$C = \begin{bmatrix} 5 & 4 & 2 \\ 7 & 1 & 7 \\ 2 & 9 & 5 \end{bmatrix}$$

وتسمى هذه المصفوفة ٣ × ٣ أى ٣ صفوف و ٣ أعمده ويتم الاشارة الى عناصر المصفوفة برقم الصف ورقم العمود الذي يقع عندهما العنصر كما يلى.

العنصر C [0] [0] يساوي ٥

العنصر C [0] [1] يساوى ٤

العنصر C [0] [2] يساوى ٢

العنصر C [2] [1] يساوى ٩

وهكذا ..

والخلاصة أن المصفوفة هى مجموعه من العناصر سواء ذات بعد واحد أو بعدين بشرط أن تكون جميع العناصر من نوع واحد وفيما يلى سنوضح كيفية الاعلان عن المصفوفة وكيفية التعامل مع عناصرها

المصفوفة ذات البعد الواحد

البرنامج الموجود بالشكل رقم (١-٦) يوضح التعامل مع المصفوفة ذات البعد الواحد وفيه يتم الاعلان عن المصفوفة واستقبال عناصر المصفوفة من المستخدم واضافة قيمة صحيحة الى كل عنصر من عناصر المصفوفة ثم طباعة عناصر المصفوفة كما يتضح ذلك من نتيجة التنفيذ

```
0:  /*Program Name CS6_1.C*/
1:  #include<stdio.h>
2:  main ()
3:      {
4:       int A[10] ;
5:       int  i;
6:       for  (i=0;i<10;i++)
7:           {
8:             printf (" \n A[%d]=",i);
9:             scanf("%d",&A[i]);
```

```
10:            A[i]= A[i]+5;
11:        }
12:     for (i=0; i<10; i++)
13:         printf (" \n A[%d]=%d",i,A[i]);
14:   } /* main () function ---- */
```

شكل رقم (١-٦) التعامل مع مصفوفة ذات بعد واحد

وعند تنفيذ هذا البرنامج نحصل على النتيجة التالية :

```
A[0]=5
A[1]=7
A[2]=56
.......
......
A[0]=10
A[1]=12
A[2]=61
............
............
```

وعن هذا البرنامج نوضح مايلي :

- في السطر رقم ٤ اعلان عن مصفوفة عدد عناصرها ١٠ عناصر ونوع هذه العناصر int واسم هذه المصفوفة هو الحرف A وطريقة الاعلان عن المصفوفة بسيطة كما لو كنت تعلن عن متغير واحد ولكنك تضيف عدد عناصر المصفوفة إلى هذا المتغير .
- في السطر رقم ٦ إستخدمنا دوارة for لاستقبال عناصر المصفوفة ولاحظ استخدام دوارة for مع المصفوفة فاذا لم تستعمل دوارة for وكان عدد عناصر المصفوفة ١٠ عناصر فلابد من كتابة السطرين التاليين عشرة مرات لاستقبال عناصر المصفوفة

```
printf ("\n A[0] =" );
scanf("%d",& A[0] );
```

وهكذا – عشرة مرات – وهذا غير معقول لذلك نستخدم دوارة for كما يلى :

```
for (i=0;i<10;i++)
   {
     printf("\n A[%d]=",i)
     scanf("%d,& A[i] );
   }
```

لابد من استعمال الدوارة for مع المصفوفات

- فى السطر رقم ٨ دالة ( ) printf تطبع العبارة = A[0]
- فى السطر رقم ٩ دالة ()scanf تستقبل الرقم الذى يدخله المستخدم وتخزنه فى عنصر المصفوفة A[i].

ونظرا لان المتغير i يبدأ بالقيمة صفر فان القيمة المدخلة تخزن فى العنصر الأول من المصفوفة والمشار اليه بالصورة A [0] ثم تزداد قيمة i وبالتالى تتوالى عناصر المصفوفة و تمتلىء بالترتيب

يشار لأول عنصر فى المصفوفة بالرقم صفر هكذا A[0].

فى السطر رقم ١٢ دوارة for أخرى لطباعة عناصر المصفوفة بعد اضافة الرقم ٥ الى كل عنصر

اعطاء قيم ابتدائيه لعناصر المصفوفة

من الممكن الاعلان عن المتغير واعطائه قيمة ابتدائية بالشكل ا لاتى

```
int A=5;
```

وهذا اعلان عن متغير صحيح وفى نفس الوقت اعطاءه قيمة ابتدائية

وبنفس الاسلوب يمكن الاعلان عن المصفوفة واعطائها قيم ابتدائية كما يلى

```
int A[3] = {5,7,9} ;
char name [10] = { 'c', 'b' , 't' , 'r', -----} ;
```

وهذا معناه اعطاء قيم ابتدائية لعناصر المصفوفة وهو الافضل كلما استطعت ذلك حتى لا يقوم البرنامج بتخزين قيم عشوائية من الذاكرة فى عناصر المصفوفة وحتى لاتطبع قيم ليس لها معنى

المصفوفة الغير محددة العدد

المقصود بها هو عدم تحديد عدد العناصر فى حالة الاعلان وتأخذ الصورة الاتية

```
int A[]={3,4,5} ;
```

أو

```
char name [ ]= " abdef ";
```

وتحديد عدد عناصر هذه المصفوفة فى هذه الحالة يتم من خلال المترجم عن طريق عد العناصر فى الطرف الايمن وحجز مصفوفة بهذا العدد

فمثلا ; int A[]={3,4,5} : معناه أن المصفوفة [ ] A عدد عناصرها ٣ عناصر وهكذا

وهذا لايصلح الا اذا كنت ستعطى عناصر المصفوفة قيم ابتدائية ولكن لا يصح أن تعلن عن مصفوفة غير محددة العدد ثم تستعملها فى استقبال قيم من المستخدم فمثلا لايصح أن تقول [ ]int a ثم تستقبل عناصر المصفوفة a من المستخدم.

مثال

يقوم البرنامج الموجود بالشكل (٢-٦) باستقبال الاسم ثم يطبعه بحيث تكون الحروف معكوسه كما يظهر ذلك من نتيجة التنفيذ.

```
/*Program Name CS6_2.C*/
#include <stdio.h>
main ( )
```

```
3:     {
4:       int  i, c=0;
5:       char name [20] ;
6:       clrscr ( );
7:       printf (" \n Enter your name :")
8:       while (name [c]= getche ( ) ) !=' \r' )
9:            c++ ;
10:      printf("\n your name in reverse is :");
11:      for (i=c ;i>= 0; i–)
12:           printf ("%c", name [i] );
13   }
```

شكل رقم ٢-٦ استقبال اسم وعكس حروفه

وعن هذا البرنامج نوضح مايلي :

في السطر رقم ٥ اعلان عن مصفوفة حروف عدد عناصرها ٢٠ كحد أقصى ويشتمل السطر رقم ٨ على الدالة getche() التي تستقبل حرف وتخزنه في عنصر المصفوفة name[c] و c لها قيمة ابتدائية صفر اى تخزنه في أول عنصر وتختبر جملة while هذا الحرف فاذا لم يكن هو مفتاح الادخال تستمر في تنفيذ الجمل التالية لها

السطر رقم ٩ يزيد قيمة العداد c بمقدار واحد بالصورة c++ ويظل البرنامج يستقبل حرف ويزيد قيمة العداد c بمقدار واحد كلما كتب المستخدم حرفا وطالما لم يضغط المستخدم على مفتاح الادخال Enter

لاحظ أن الدالة getche() لاتحتاج الى ضغط مفتاح Enter لذلك يستمر الاستقبال في سطر واحد كعبارة حرفية.

- في السطر رقم ١١ الدوارة for تبدأ من آخر عنصر تم ادخاله وهو آخر قيمة للمتغير c وتنتهي عند أول عنصر ورقمه صفر.

- في السطر رقم ١٢ يتم طباعة عناصر المصفوفة بالعكس وذلك لاننا بدأنا من أخر عنصر ادخل حتى أول عنصر

وعند تنفيذ البرنامج تحصل على النتيجة التالية :

```
Enter your name:SAMY
your name in reverse is:YMAS
```

الأشكال المختلفة لتراكيب arrays فى لغة ++C:

سبق وذكرنا أن البيانات داخل array عادة ما تكون من النوع البسيط مثل integer أو float أو char . والآن نشير إلى إمكانية وضع بيانات من النوع التركيبى فى array تعرف فى هذه الحالة باسم array متعددة الأبعاد (multidimensional array) . ويعتبر هذا التركيب من الأنواع التركيبية لأن فى هذه الحالة تكون مؤلفة arrays فى هذه الحالة تكون مؤلفة فى داخلها من مجموعة من arrays وليست array واحدة . والآن ، سنعرض لأحد هذه الأنواع وهو : (array ثنائية البعد) two-dimensional array .

كيفية معالجة البيانات التركيبية فى array ثنائية البعد:

تأخذ البيانات المخزنة فى array ثنائية البعد شكل شبكة من الصفوف والأعمدة المتداخلة ، يوضحها الشكل القادم : فلو فرضنا أن الإحداثى الرأسى هو عدد الصفوف المدون بها الأيام الخمس الأولى من افتتاح أحد المحال التجارية ، وكان الإحداثى الأفقى يمثل عدد الأعمدة المدون بها الأسابيع والشهور ، فإن الشكل التالى يمثل المبيعات اليومية لهذا المحل التجارى فى مدة أقصاها أربعة أسابيع .

		Column Index			
		1	2	3	4
Row Index	1	91	22	37	40
	2	58	63	99	35
	3	77	81	23	62
	4	92	15	33	29
	5	102	73	62	64

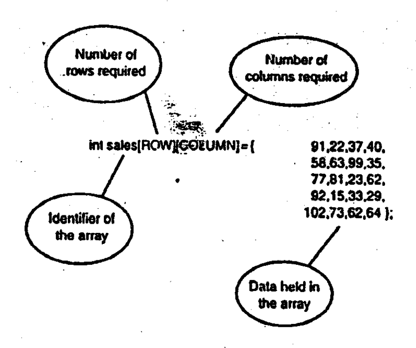
كشف بمبيعات هذا المحل التجارى، تمثله مجموعة من الصفوف والأعمدة المتداخلة:

كيفية استدعاء العناصر فى array ثنائية البعد:

يتضح من الشكل السابق أنه يلزم لتحديد عنصر من عناصر البيانات فى هذا النوع من تراكيب arrays أن نذكر رقم الصف والعمود الموجود فيهما . فعلى سبيل المثال لو كان العنصر المراد استدعائه من هذا التركيب يكون برقم الصف "3"، ورقم العمود "2"، فمعنى ذلك أن هذا العنصر موجود فى ٣ رأسى و ٢ أفقى، ويحمل القيمة [18] .

كيفية تحديد العنصر فى array ثنائية البعد:

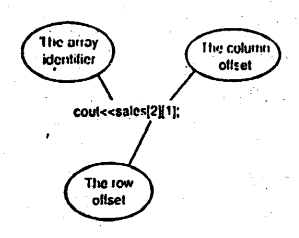
يمكن تخزين البيانات الخاصة بمبيعات المحال التجارية باستخدام array ثنائية البعد وذلك من خلال تحديد المكان المراد إدخال البيانات إليه فى التركيب. ولأن تركيب array ثنائية البعد يكون مشتملاً على إحداثيين، فسيتم تحديد العنصر داخل array بوضع رقمين يشيران إلى موضع العنصر المراد إدخال البيانات إليه. وسيكون الرقم الأول هو رقم الصف الذى يشير إلى الأيام، والرقم الثانى هو رقم العمود الذى يشير إلى الأسابيع. ويوضح الشكل التالى تركيب الجملة البرمجية فى هذه الحالة:



عملية تعريف array ثنائية البعد:

كيفية تحديد عنوان العناصر فى Array ثنائية البعد:

والآن، بعدما تعرفنا على كيفية تحديد العنصر المراد إدخال البيانات إليه فى array ثنائية البعد '' اء رقم الصف ثم رقم العمود الموجود فيهما هذا العنصر. وبعد ما تعرفنا أيضًا على كيفية عرض بيانات أحد العناصر على الشاشة باستخدام أمر cout. نأتى الآن إلى كيفية تحديد عنوان عنصر فى هذا التركيب. فيلزم لاستدعاء العنصر من array ثنائية البعد أن نذكر الإسم المحدد لها متبوعًا بعنوان الصف والعمود الموجود فيهما موضوعين بين قوسين. ويجب على المستخدم أن يتذكر دائمًا أن عنوان الصف الأول والعمود الأول فى array عادة ما يكون 0. ويجب أن يتذكر أيضاً أن هذا العنوان عادة ما يكون أقل من الرقم الفهرسى لهذا العنصر فى التركيب بفارق واحد صحيح. وبتطبيق ذلك على المثال التالى، فإن العددين 2، 1 يشيران إلى عنوان العنصر المطلوب استدعاءه من التركيب، ويكون تركيب الجملة البرمجية بهذا الشكل:



كيفية استدعاء البيانات من إحدى العناصر
فى array ثنائية البعد باستخدام أمر cout:



كيفية تنسيق البيانات فى source code البرنامج:

يمكن استخدام هذا النوع من تراكيب arrays فى كتابة البيانات فى source code فى شكل صفوف وأعمدة. وليس بالضرورة أن يفعل المستخدم ذلك. فكل ما يجب عليه فعله هو أن يكتب البيانات فى سطر واحد، وأن يتوالى إدخال عناصر البيانات واحداً بعد الآخر.

مثال تطبيقى لكيفية استخدام array ثنائية البعد:

والآن، يمكنك تشغيل البرنامج القادم الموضح فى مثال (١٠-١). وسيتم فيه كتابة مجموعة عناصر البيانات التى سيتم تخزينها فى array ثنائية البعد. ويفضل استخدام جملة nested for loop فى عملية الإدخال هذه لأنه لابد من وجود عنوانيين للعنصر فى هذا التركيب. وستكون خطوات التنفيذ كما يلى:

- وضع القيمة المبدئية [0] باستخدام جملة for loop الخارجية.
- المرور على جميع عناوين الأعمدة ابتداً من 0 وحتى 3، ويكون بذلك قد تم الدخول إلى nested for loop.

ه رفع قيمة المتغير السابق إلى [1] ، وبذلك يكون قد تم الدخول ثانياً إلى جملة for loop الخارجية .

ه المرور على جميع الأعمدة التى تحمل العناوين من 0 وحتى 3 للدخول ثانياً إلى جملة الـ nested for loop .

شغل البرنامج بالخطوات السابقة ، وستظهر أمامك على الشاشة المخرجات التالية :

```
91 22 37 40

58 63 99 35

77 81 23 62

92 15 33 29

102 73 62 64
```

مثال (١-١): نموذج لبرنامج يستخدم array ثنائية البعد:

```
#include <iostream.h>
#define ROW 5     // FIVE WORKING DAYS
#define COLUMN 4   // FOUR WEEKS IN MONTH
main()
{  int sales[ROW][COLUMN] = { 91,22,37,40,
            58,63,99,35,
            77,81,23,62,
            92,15,33,29,
            102,73,62,64};
  int down;   // go down the rows
  int across;   // go across the columns

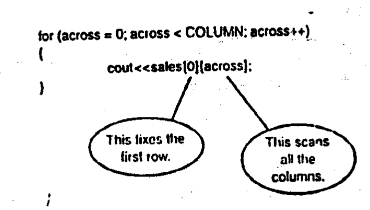
  cout.setf(ios::right);
  cout << endl;
  for (down = 0; down < ROW; down++)
  {
   for (across = 0; across < COLUMN; across++)
   {
    cout.width(4);
    cout << sales[down][across];
```

```
}
  cout << endl << endl;
  }
 return(0);
}
```

ويلاحظ فى البرنامج السابق أنه قد تم عرض جميع البيانات داخل array فى شكل مجموعة متداخلة من الصفوف والأعمدة. ولكن ماذا لو أراد المستخدم استدعاء بعض بيانات التركيب وليس جميعها؟ .

مثال تطبيقى لاستدعاء بيانات أحد الصفوف فى array ثنائية البعد:

وبفرض أنك أردت عمل كشفًا بمبيعات يوم العمل الأول من كل أسبوع فى أحد المحال التجارية ، فإنك ستحتاج إلى استخدام هذا النوع من تراكيب array . وسيكون ذلك بتحديد عنوان الصف الأول منها الذى يمثل الأيام ، والذى دائمًا ما يكون 0 . ويتم الاحتفاظ بهذه القيمة ثابتة لا تتغير ، وقد تستخدم جملة for loop بهذه الكيفية :



استخدام جملة for loop لعرض صف واحد فى array ثنائية البعد

وستظهر مبيعات اليوم الأول (الإثنين) بعد تشغيل البرنامج الموضح فى مثال (١٠-٢) بهذا الشكل :

19 22 37 40



- حاول ذلك -

حاول تغيير العنوان السابق إلى أى من العناوين التالية (١، ٢، ٣، ٤)، ولاحظ النتيجة فى كل مرة.

مثال (١-٢): نموذج لبرنامج يستخدم array ثنائية البعد لعرض محتويات صف معين:

```
#include <iostream.h>

#define ROW 5
#define COLUMN 4

main()
{
  int sales[ROW][COLUMN] = {  91,22,37,40,
                              58,63,99,35,
                              77,81,23,62,
                              92,15,33,29,
                              102,73,62,64};
  int across;   // This is the columns

  cout.setf(ios::right);
  cout << endl << "A Day In The Month";
  cout << endl << endl;
  for (across = 0; across < COLUMN; across++)
  {
   cout.width(4);
   cout << sales[0][across];
  }
  cout << endl << endl;
  return(0);
}
```

مثال تطبيقي لاستدعاء أحد الأعمدة في array ثنائية البعد:

وبفرض أنك أردت هذه المرة أن تعد كشفًا بمبيعات الأسبوع الثاني من أحد الشهور في إحدى المحال التجارية، يمكنك استخدام array ثنائية البعد لتنفيذ ذلك. ويكون هذا بتحديد عنوان الإحداثي الثاني الذي يكون في هذه الحالة [1]، ويتم نتيجة لذلك فحص بيانات هذا العمود بالكيفية التالية:

```
for (down = 0; down < ROW; down++)
{
        cout<<sales[down][1];
}
```

This scans all the rows.

This fixes the second column.

استخدام جملة for loop لعرض عمود واحد فى array ثنائية البعد .

وبعد إجراء الخطوات السابقة، ستظهر بيانات العمود بالشكل التالى :

22
63
81
15
73

مثال (١-٢): نموذج لاستخدام array ثنائية البعد فى فحص محتويات أحد الأعمدة.

```
#include <iostream.h>
#define ROW 5
#define COLUMN 4

main()
{
int sales[ROW][COLUMN] = {  91,22,37,40,
 58,63,99,35,
                              77,81,23,62,
                              92,15,33,29,
                              102,73,62,64};

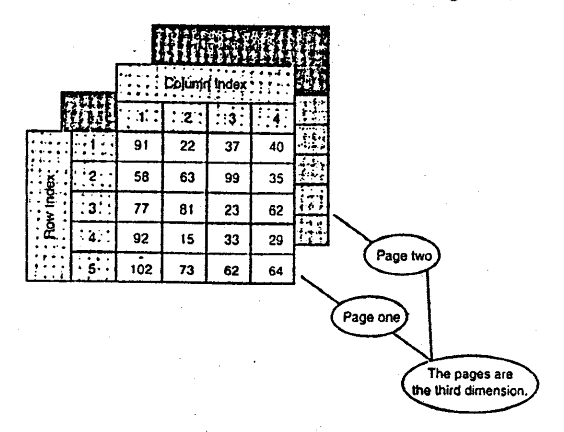
 int down;   // This is the row

 cout.setf(ios::right);
 cout << endl << "A Week In The Month Observed.";
 cout << endl << endl;
 for (down = 0; down < ROW; down++)
 {
      cout.width(4);
 cout << sales[down][1] << endl;
```

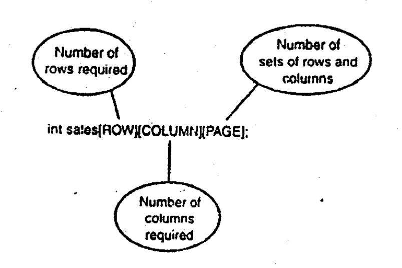
```
}
cout << endl << endl;
return(0);
}
```

تراكيب arrays ثلاثية البعد:

يشمل هذا النوع من تراكيب arrays على ثلاث إحداثيات هى : الصف والعمود والصفحة . ويكون شكل البيانات فيها على هذا النحو :



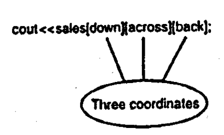
أسلوب تنسيق البيانات فى array ثلاثية البعد:



كيفية تعريف array ثلاثية البعد بتحديد رقم الصف والعمود والصفحة

كيفية استدعاء عناصر البيانات فى array ثلاثية البعد:

لابد فى هذا النوع من تراكيب arrays أن يحدد المستخدم ثلاث إحداثيات لها، وبالشكل التالى:



الإحداثيات الثلاثة فى array ثلاثية الأبعاد

كيفية تكوين array ثلاثية البعد:

ستلاحظ فى السطور القليلة التالية أن طريقة وضع قيمة فى array من هذا النوع أو ما كان يعرف باسم initialization قد اختلفت، ويرجع ذلك إلى سببين:

- الحاجة إلى إدخال قيمة معينة إلى إحدى عناصر التركيب بالكيفية نفسها المستخدمة لإدخال قيم إلى المتغيرات فى خلال runtime .
- الرغبة فى توضيح أسلوب in-line الممكن استخدامه لكتابة الكود.



هل يمكن تكوين المزيد من الصفحات:

بالطبع نعم. يمكنك زيادة عدد الصفحات فى array بزيادة قيمة المتغير PAGE.

وستكون المخرجات على الشاشة فى مثال (١٠-٤) الذى يوضح مبيعات المحل التجارى فى مدة أقصاها شهرين على النحو التالى:

```
91 22 37 40

58 63 99 35
```

77 81 23 62

92 15 33 29

102 73 62 64

12 23 27 42

59 83 79 75

77 71 63 72

110 105 103 109

102 83 61 63

مثال (٤-١٠): نموذج لبرنامج يستخدم array ثلاثية البعد:

```
#include <iostream.h>

#define ROW 5          // FIVE WORKING DAYS
#define COLUMN 4       // FOUR WEEKS IN MONTH
#define PAGE 2         // MONTHS 1 AND 2

main()
{
 int sales[ROW][COLUMN][PAGE];
 int down;
 int across;
 int back = 0;

 sales[0][0][0] = 91;
 sales[0][1][0] = 22;
 sales[0][2][0] = 37;
 sales[0][3][0] = 40;

 sales[1][0][0] = 58;
 sales[1][1][0] = 63;
 sales[1][2][0] = 99;
```

```
sales[1][3][0] = 35;

sales[2][0][0] = 77;
sales[2][1][0] = 81;
sales[2][2][0] = 23;
sales[2][3][0] = 62;

sales[3][0][0] = 92;
sales[3][1][0] = 15;
sales[3][2][0] = 33;
sales[3][3][0] = 29;

sales[4][0][0] = 102;
sales[4][1][0] = 73;
sales[4][2][0] = 62;
sales[4][3][0] = 64;

sales[0][0][1] = 12;
sales[0][1][1] = 23;
sales[0][2][1] = 27;
sales[0][3][1] = 42;

sales[1][0][1] = 59;
sales[1][1][1] = 83;
sales[1][2][1] = 79;
sales[1][3][1] = 75;

sales[2][0][1] = 77;
sales[2][1][1] = 7[illegible];
[illegible]
sales[2][[illegible]][1] = [illegible];

sales[3][0][1] = 110;
sales[3][1][1] = 105;
sales[3][2][1] = 10[illegible];
sales[3][3][1] = 109;

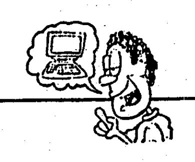
sales[4][0][1] = 102;
```

```
sales[4][1][1] = 83;
 sales[4][2][1] = 61;
 sales[4][3][1] = 63;

 cout.setf(ios::right);
 cout << endl;
 for (down = 0; down < ROW; down++)
 {
  for (across = 0; across < COLUMN; across++)
  {
   cout.width(4);
   cout << sales[down][across][back];
  }
  cout << endl << endl;
 }
 back++;
 cout << endl << endl;
 for (down = 0; down < ROW; down++)
 {
  for (across = 0; across < COLUMN; across++)
  {
   cout.width(4);
   cout << sales[down][across][back];
  }
  cout << endl << endl;
 }
 return(0);
}
```

مفهوم (String) Array of Characters :

ذكرنا فى الفصل السابق أن مصطلح array of characters هو نفسه مصطلح string . ويعنى ذلك أن string عبارة عن array أحادية البعد من نوع char . وعلى ذلك ، فإن تركيب array المشتمل على عدد من بيانات string هو فى الواقع array ثنائية البعد من نوع char .



أسلوب in-Line:

سبق واستخدمنا هذا الأسلوب فى الفصل السابق، وهو واحد من الأسلوبين اللذين يتم بهما تكوين array.

فى الأسلوب الأول يتم تعريف array، وتكوينها فى سطر واحد من الكود مثل أى متغير من المتغيرات:

```
int demo [4] = {33,75,12,92} ;
```

وهذا ما يطلق عليه أسلوب in-line فى الإدخال.

أما الأسلوب الثانى فهو الأسلوب المطول الذى يتم فيه تحديد كل عنوان على حده مع إضافة القيمة المطلوب إدخالها إليه:

```
deme[0] = 33 ;
deme[1] = 75 ;
deme[2] = 12 ;
deme[3] = 92 ;
```

ويقدم مثال (١٠-٥) نموذج برنامج يستخدم array من نوع char فى عرض بيانات بأسماء عدد من المشاهير. ويلاحظ أن عملية عرض البيانات فى هذه المرة لم يختلف عن الكيفية السابق العرض لها.

ويمكنك المرور على جميع الص فى التركيب باستخدام متغير index. ويلاحظ أنها نفس الكيفية المستخدمة فى الأمثلة السابقة، ولكن مع اختلاف أنه قد تم تحديد إحداثى واحد فى هذا المثال. ولو كان الإحداثى المطلوب عرضه - مثلاً - عنوانه 0، فإن compiler الجهاز يفهم على الفور أن الإحداثى المطلوب هو الإحداثى الأول فى التركيب. وعلى ذلك، فلابد من وضع الاسم الكامل لكل إحداثى.

وبعد كتابة البرنامج الموضح فى مثال (١٠-٥) وتشغيله، سيكون شكل المخرجات على الشاشة على هذا النحو:

```
DAVY CROCKETT
JOHN LENNON
MOHAMMED ALI
WILLIAM WALLACE
```



هل هناك أشكال أخرى من تراكيب array؟:

يمكن أن يكون لتركيب array عدداً كبيراً من الأبعاد يصل إلى أربعة أو خمسة أو ستة. ولكن لابد من توخى الحذر عند إنشاء مثل هذا النوع من التراكيب لأنه عادة ما يكون معقداً بالقدر الذى يسمح بالوقوع فى العديد من الأخطاء.

مثال (١٠-٥): نموذج لبرنامج يستخدم array من نوع char:

```
#include <iostream.h>

#define MAXNAMES 4        // FOUR NAMES IN ARRAY
#define MAXLENGTH 20      // NOT MORE THAN 19
CHARACTERS

main()
{
  char name[MAXNAMES][MAXLENGTH] =  {
                                       "DAVY CROCKETT",
                                       "JOHN LENNON",
                                       "MOHAMMED ALI",
                                       "WILLIAM WALLACE"};
  int index;

  for (index = 0; index < MAXNAMES; index++)
  {
   cout << endl << name[index];
  }
  cout << endl << endl;
  return(0);
}
```

الدروس المستفادة:

array متعددة الأبعاد عبارة عن array واحدة مؤلفة من عدة arrays أخرى؛ تماماً مثل نوع البيانات الموجودة داخلها والتى قد تكون من نوع integer - مثلاً - الذى ينتمى لبيانات النوع البسيط، أو من نوع struct الذى ينتمى لبيانات النوع التركيبى.

تتكون array ثنائية البعد من إحداثيين: الصفوف والأعمدة. ويمكن الإشارة إليها بالإحداثى x والإحداثى y. ويتم تخزين البيانات فى مواضع تداخل هذين الإحداثيين.



نقطة أخيرة:

. تم تعريف string على أنها array مكونة من عشرين عنصر، فإن معنى ذلك، أن arra لا تحمل سوى تسعة عشر عنصراً من نوع character. وذلك لأن العنصر ثانى سيتم حفظه فى شكل آخر من أشكال characters يطلق عليه اسم Null characte. ويستخدم هذا النوع فى لغة ++C لتحديد المدى الذى تنتهى عنده strin، وذلك من أجل تيسير عمليات التعامل مع strings فى البرنامج.

- string عبارة عن array أحادية البعد من نوع char. أما مجموعة strings، فهى عبارة عن array ثنائية البعد من نوع char.
- يستخدم Null char فى تركيب array لتحديد المدى الذى تنتهى عنده جملة string، الأمر الذى يمكن معه معالجة بيانات التركيب بسهولة.
- يمكن من الناحية النظرية أن تتكون array من عدد لا نهائى من الأبعاد. ولكن من الناحية العملية يصعب القيام بذلك لأنه يستلزم عندئذ إضافة إحداثى جديد لكل بعد يتم إضافته إليها.

كيفية دمج أنواع البيانات باستخدام Parallel Array:

[illegible] أن جميع عناصر array يجب أن تكون من نفس نوع البيانات. ولكن ماذا يحدث إذا أردت تمثيل مزيج من أنواع البيانات؟ يمكن تحقيق ذلك باستخدام اثنين أو أكثر من parallel arrays التى تستطيع أن تصل إليها باستخدام فهرس عام. افترض أنك أردت أن ترسم جدولا يشتمل على الكواكب الموجودة فى النظام الشمسى وبعدها عن الشمس. ستكون أسماء الكواكب من نوع string، وسيكون بعدها عن الشمس من نوع integer. لذلك يجب عليك إنشاء اثنين من parallel arrays مستخدمًا المعلومات الموضحة فى جدول الكواكب.

يوجد تسع كواكب معروفة، لذلك يجب أن تشتمل arrays على حيز لتسعة عناصر. ويتم توضيح هذا فى بداية مثال (١١-١) باستخدام #define MAX 9. ولا تنس أنه يتم استخدام مدى خاص بـ array التى تبدأ من 0 وتنتهى عند 8، وتشتمل على تسع كواكب.

Index	Planet	Distance
1	Mercury	58
2	Venus	108
3	Earth	150
4	Mars	228
5	Jupiter	778
6	Saturn	1427
7	Uranus	2869
8	Neptune	4498
9	Pluto	5900

البيانات الخاصة بالكواكب التسعة المعروفة.

مثال (١١-١): عرض Parallel Arrays

```
#include <iostream.h>
#include <string.h>

#define MAX 9            // There are 9 known planets
#define LENGTH 8         // No name is longer than 7
                         // characters
#define TAB '\t'         // Define a tab character

main()
{
```

```
char planet[MAX][LENGTH];    // planet names
int  distance[MAX];          // distance from sun
int  index;

// INITIALIZE THE TWO ARRAYS
strcpy(planet[0],"MERCURY");
distance[0] = 58;
strcpy(planet[1],"VENUS");
distance[1] = 108;
strcpy(planet[2],"EARTH");
distance[2] = 150;
strcpy(planet[3],"MARS");
distance[3] = 228;
strcpy(planet[4],"JUPITER");
distance[4] = 778;
strcpy(planet[5],"SATURN");
distance[5] = 1427;
strcpy(planet[6],"URANUS");
distance[6] = 2869;
strcpy(planet[7],"NEPTUNE");
distance[7] = 4498;
strcpy(planet[8],"PLUTO");
distance[8] = 5900;

cout << TAB << TAB << "PLANET" << TAB << TAB
        << "DISTANCE" << endl << endl;
for (index = 0; index < MAX; index++)
{
   cout << TAB << TAB << planet[index];
   cout << TAB << TAB << distance[index] << endl;
}
cout << endl;
return(0);
}
```

لا يزيد عدد أحرف اسم كل كوكب من الكواكب عن سبعة أحرف، ولاستخدام NULL terminator (علامة إنهاء string)، يتم تحديد عدد الأحرف باستخدام #define LENGTH 8.

وداخل الجزء الرئيسى من البرنامج، يجب إنشاء اثنتين من arrays ، حيث تخصص واحدة لكل عمود. وتحتوى الـ array الأولى ثنائية الأبعاد على بيانات من نوع character (والمعروفة الآن أيضًا بأنها array من نوع string) وتحتوى الـ array الأخرى على بيانات من نوع integer.

```
char planet[MAX][LENGTH];
```

```
int distance[MAX];
```

تتمثل الخطوة الأخيرة فى تكوين arrays لحفظ أسماء الكواكب التسعة المعروفة بعدها عن الشمس. وهذه البيانات موضحة فى الجدول السابق. وعندما يتم إعداد محتويات هذين المتغيرين، يجب الالتزام بترتيب الفهرس لأن هذا يعد أمراً ضرورياً. ليس من الصحة فى شئ أن يتوافق اسم الكواكب MARS مع البعد الخاص بكوكب SATURN.

كواكب: برنامج ملائم لوكالة الفضاء NASA:

بدون جهد كبير، قمنا فى الجزء السابق بإعداد برنامج الكواكب. فلقد تم التعرف من كل على جميع مفاهيم البرمجة، والآن يتم فقط إعادة ترتيبها بشكل طفيف.

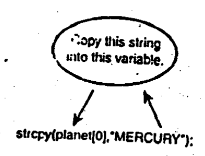
أثناء تنفيذ loop، تتزايد بيانات الفهرس قيمة واحدة فى كل مرة، لذلك تكتب أولاً البيانات الخاصة بكوكب Mercury على الشاشة. ثم تظهر البيانات التالية الخاصة بكوكب Venus فى المرة الثانية التى يتم فيها تنفيذ loop، ثم Earth، ثم Mars وهكذا. ولا يعتبر هذا جديداً للمبرمجين المتخصصين، لذلك سنضيف ميزتين جديدتين.

شاشة TAB:

تتمثل الميزة الأولى فى TAB، التى يتم استخدامها فى تنظيم بيانات الشاشة. تسبق Control characters (أحرف أو رموز التحكم) فى لغة ++C بالرمز \، بينما يقوم الحرف t بتمثيل TAB character. وعندما يتم دمج الحرف والرمز، ينتج control character الذى تم إعداده باستخدام '\t' #define TAB. وبهذا يمكن استخدام TAB فى أى مكان فى البرنامج لإدراج مسافة TAB.

ملاحظات حول String.h:

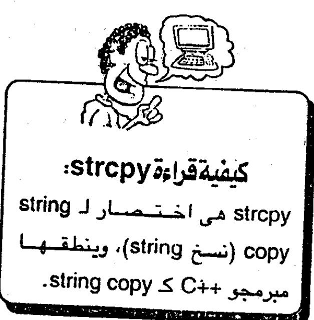
كما هو معروف تعتبر strings فى الواقع arrays من نوع character، ولا يمكن التعامل مع strings كمتغيرات عادية. حيث تكمن أكثر نقاط الاختلاف فى أنه لا يمكن تكوين string بنفس الطريقة التى يمكن بها تكوين integer أو float. وللتغلب على هذه المشكلة البسيطة، قام مطورو C++ compilers بتقديم library خاصة تسمى string.h، التى تشتمل على functions يمكنها القيام بهذه الوظائف الخاصة. وأول هذه functions هى strcpy.



استخدام أمر strcpy فى تكوين متغير من نوع string باستعمال بيانات string.

والآن يمكن تنفيذ برنامج الكواكب وملاحظة المخرجات على الشاشة : حيث يجب أن يبدو بصورة مميزة كالمخرجات الموضحة هنا :

PLANET	DISTANCE
MERCURY	58
VENUS	108
EARTH	150
MARS	228
JUPITER	778
SATURN	1427
URANUS	2871
NEPTUNE	4497
PLUTO	5914



كيفية قراءة strcpy:

strcpy هى اختصار لـ string copy (نسخ string)، وينطقها مبرمجو ++C كـ string copy.

آخر خبر. وأخيراً اكتشاف كوكب Vulcan!:

لقد اكتشف مؤخراً كوكب جديد يسمى Vulcan . يبلغ بعده عن الشمس ٨,٩٩٢ مليون كيلو متر . وقد انطلقت وكالة الفضاء NASA لتحديث قاعدة بياناتها .

لقد قام العلماء بعمل شاق . والآن يجب على المبرمجين المختصين أن يقوموا بتحديث قاعدة البيانات . ففى بداية هذا الفصل، تم تحديد عدد الكواكب باستخدام #define MAX 9 . وذلك يعنى أنه يوجد تسع كواكب . وبإضافة كوكب Vulcan ، تصبح الآن الكواكب عشرة . لذلك فكل ما يجب عمله هو تغيير عددها من تسعة إلى عشرة .

يعتبر إضافة البيانات الجديدة أمراً سهلاً . فقد أمدتنا بها وكالة الفضاء NASA .
ذلك يثبت أن الطريقة التى تم بها تكوين string تختلف عن الطريقة التى يتم بها تكوين
لمتغير البسيط .

```
strcpy(planet[9],"VULCAN");
distance[9] = 8992;
```

يعرض مثال (١١-٢) البرنامج كامل بعد إضافة كوكب Vulcan إلى القائمة .
عندما يتم تنفيذ البرنامج، سيكون Vulcan قد انضم فعلاً إلى النظام الشمسى، كما هو
يلاحظ فى الإخراج التالى :

```
PLANET      DISTANCE

MERCURY     58
VENUS       108
EARTH       150
MARS        228
JUPITER     778
SATURN      1427
URANUS      2871
NEPTUNE     4497
PLUTO       5914
VULCAN      8992
```



تفضيل المبرمجين للنسخ

عندما يتم تعديل برنامج موجود، لا يجب كتابة العمل بالكامل مرة ثانية. فالمبرمجون دائمًا يأخذون الكود الموجود، ويعدلونه، ثم يقومون بحفظه باسم مختلف. وفى هذه الحالة سنطبق فعلاً القول المأثور القديم "الوقت من ذهب". فلا يجب إضاعة الوقت فى [illegible]

مثال (۱۱-۲)، واخیراً اکتشاف کوکب Vulcan

```
#include <iostream.h>
#include <string.h>

#define MAX 10            // There are NOW 10 known planets
#define LENGTH 8          // No name is longer than 7
                          // characters
#define TAB '\t'          // Define a tab character

main()
{
   char planet[MAX][LENGTH];     // planet names
   int  distance[MAX];           // distance from sun
   int  index;

   // INITIALIZE THE TWO ARRAYS
   strcpy(planet[0],"MERCURY");
   distance[0] = 58;
   strcpy(planet[1],"VENUS");
   distance[1] = 108;
   strcpy(planet[2],"EARTH");
   distance[2] = 150;
   strcpy(planet[3],"MARS");
   distance[3] = 228;
   strcpy(planet[4],"JUPITER");
   distance[4] = 778;
   strcpy(planet[5],"SATURN");
   distance[5] = 1427;
   strcpy(planet[6],"URANUS");
   distance[6] = 2869;
   strcpy(planet[7],"NEPTUNE");
   distance[7] = 4498;
   strcpy(planet[8],"PLUTO");
   distance[8] = 5900;
   strcpy(planet[9],"VULCAN");
   distance[9] = 8992;

   cout << TAB << TAB << "PLANET" << TAB << TAB
```

```
        << "DISTANCE" << endl << endl;
    for (index = 0; index < MAX; index++)
    {
        cout << TAB << TAB << planet[index];
        cout << TAB << TAB << distance[index] << endl;
    }
    cout << endl;
    return(0);
}
```

تكبير Parallel Array:

لقد قررت وكالة الفضاء NASA تحديث قاعدة بياناتها للمرة الثانية. وأصدرت المواصفة التالية:

لقد تقرر إضافة عدد الأقمار التابعة لكل كوكب إلى قاعدة البيانات. وبإصدار المعلومات الإضافية، يتم إنشاء parallel arrays الضرورية بنفس الطريقة المستخدمة فى قاعدة البيانات القديمة. ويجب طباعة المعلومات الزائدة على الشاشة بطريقة مماثلة لتلك التى تم بها طباعة المعلومات الموجودة.

Index	Planet	Moons	Year	Distance
1	Mercury	0	0.024	58
2	Venus	0	0.625	108
3	Earth	1	1	150
4	Mars	2	1.91	228
5	Jupiter	16	12	778
6	Saturn	18	29.9	1427
7	Uranus	15	85.24	2869
8	Neptune	8	167.19	4498
9	Pluto	1	251.29	5900

المعلومات الإضافية الخاصة ببرنامج الكواكب.

وبما أن البيانات الجديدة تشتمل على أربع أعمدة، مرتبطة ببعضها بواسطة فهرس عام، فمن اللائق أن يتم استخدام أربع arrays بشكل متوازى.

```
char planet[MAX][LENGTH];
int moons[MAX];
float year[MAX];
int distance[MAX];
```

تكمن المشكلة الكبرى فى هذا النوع من البرامج في حجم المساحة المطلوب حجزها فى الذاكرة. لذلك يفضل القطع والإضافة أينما كان ممكنًا. وفيما يلى يتمثل البرنامج.

مثال (١١-٢): عرض Parrallel Arrays الأربعة.

```
#include <iostream.h>
#include <string.h>

#define MAX 9          // There are 9 known planets
#define LENGTH 8       // No name is longer than 7
                       // characters
#define TAB '\t'       // Define a tab character

main()
{
   char planet[MAX][LENGTH];      // planet names
   int moons[MAX];                // number of moons
   float year[MAX];               // length of year
   int distance[MAX];             // distance from sun
   int index;

   // INITIALIZE THE FOUR ARRAYS
   strcpy(planet[0],"MERCURY");
   moons[0] = 0;
   year[0] = 0.24;
   distance[0] = 58;
   strcpy(planet[1],"VENUS");
   moons[1] = 0;
   year[1] = 0.625;
   distance[1] = 108;
   strcpy(planet[2],"EARTH");
   moons[2] = 1;
   year[2] = 1;
   distance[2] = 150;
```

```
strcpy(planet[3],"MARS");
moons[3] = 2;
year[3] = 1.91;
distance[3] = 228;

cout << TAB << TAB << "PLANET"
    << TAB << TAB << "MOONS"
    << TAB << TAB << "YEAR"
    << TAB << TAB << "DISTANCE"<< endl << endl;
for (index = 0; index < 4; index++)
{
   cout << TAB << TAB << planet[index];
   cout << TAB << TAB << moons[index];
   cout << TAB << TAB << year[index];
   cout << TAB << TAB << distance[index] << endl;
}
cout << endl;
return(0);
}
```



فقد خمس كواكب

لقد تم إعداد الأربع arrays الخاصة بالكواكب التسعة ولكن أضيفت البيانات الخاصة بالأربعة الأولى فقط. ويعتبر ذلك أسلوباً صحيحاً لاختبار البرنامج. وعندما يتم التأكد من أنه يعمل بطريقة سليمة، يجب نقله إلى المبرمج لملء باقي arrays .

أهم الدروس المستفادة:

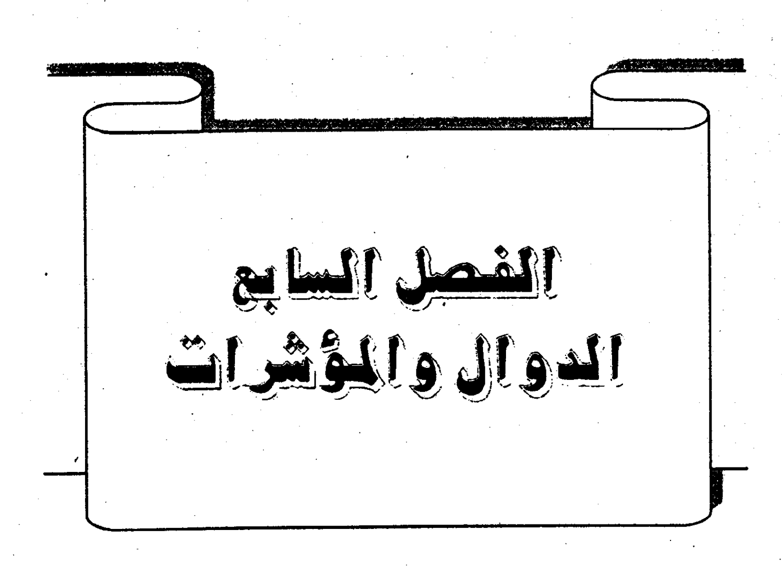
- يجب أن تكون جميع عناصر array من نفس نوع البيانات، ولكن يمكن التغلب على ذلك باستخدام اثنين أو أكثر من parallel arrays التي يمكن الوصول إليها باستخدام فهرس عام.



الانتقال من float إلى double

فى برنامج parallel array المكبرة والموجود فى مثال (١١-٣)، تم استخدام نوع جديد من البيانات يسمى double. وفى الواقع يعتبر هذا النوع من البيانات float كبير جداً ويحتاج إليه هذا البرنامج لأن الثوابت فى لغة ++C باستخدامها الفاصلة العشرية تعتبر أيضاً doubles. وتستخدم الثوابت فى حفظ طول العام ويتم نسخها من خلال البرنامج. وهى لا تسبب مشكلة فى حقيقة الأمر ولكنها تقدم تحذير عندما يتم إجراء عملية Compilation للبرنامج.

- يستخدم البرنامج المصمم بلغة ++C اثنتين من parallel arrays بتعريف اثنتين من arrays وكتابتهما معًا باستخدام فهرس عام.
- يتم تكوين عناصر parallel array بنفس الطريقة التى يتم بها تكوين عناصر array . ومع ذلك يجب الاهتمام باستخدام فهرس عام.
- يمكن الوصول للعناصر فى parallel arrays بنفس الطريقة التى يمكن بها الوصول للعناصر فى array . ومع ذلك يجب الاهتمام باستخدام فهرس عام.
- يمكن وضع العناصر الإضافية فى parallel array بنفس الطريقة التى يتم بها وضع العناصر فى array . ومع ذلك يجب الاهتمام باستخدام فهرس عام.
- تسبق Control characters فى لغة ++C بالرمز \، بينما يقوم الحرف t بتمثيل TAB character. وعندما يتم دمج الحرف والرمز، ينتج Control characters يمكن إعداده باستخدام '#define TAB'\t. وبهذا يمكن استخدام الكلمة TAB فى أى مكان فى البرنامج لإدراج مسافة TAB.
- لا يمكن استخدام علامة التساوى لنسخ string ووضعها بداخل متغير من نوع string، لأن string هى بالفعل عبارة عن array من نوع character. لذلك يجب استخدام أمر strcpy الموجود فى string.h library.
- لكتابة برنامج بلغة ++C الذى يستخدم parallel arrays عديدة، يجب ببساطة زيادة عدد arrays التى يمكن الوصول إليها باستخدام فهرس عام.


الفصل السابع
الدوال والمؤشرات



الفصل السابع
الدوال والمؤشرات

لا يمكن تصور أن يقوم أي مبرمج بكتابة برنامجه بكل تفاصيله – من الألف إلى الياء – بالمبـادئ الأوليـة لعمليات البرمجة ، فمن أهم مميزات لغات البرمجة خاصة الحديث منها أنها توفر مجموعة من الوظـائف الـتي يمكن استدعائها ويسمى كل منها دالة Functions.

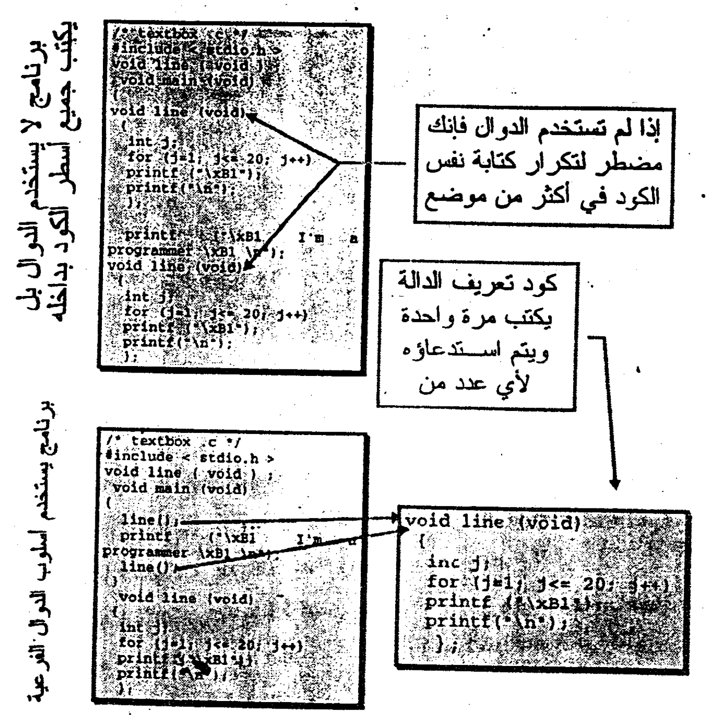
ما هي وظيفة الدالة !!؟

كما ذكرنا من قبل فإن الدالة هي في حقيقتها برنامج يتم استدعاؤه لتنفيذ مهمة بدلاً من كتابتها من جديد. والدالة Function في لغة C تماثل البرنـامج الفرعـي Sub Routine في لغـةBasic أو الإجـراءات Procedures في لغة Pascal أو Visual Basic وتتمثل فائدة استخدام الدالة في :

1 – تلافي تكرار كتابة Code واحد عدة مرات في عدة مواضع . وهو السبب الرئيسـي لاسـتخدام الدوال ، فعلى فرض أنك تكتب برنامجاً لمهمة ما ، يستلزم في جزء منه كتابة بعض الخطوات لحسـاب المتوسط الحسابي لبعض القيم الرقمية ، فإن دعت الحاجة بعد ذلك إلى حساب المتوسط الحسابي لمجموعة قيم أخرى في مكان آخر من البرنامج سيكون – في حالة عدم الاستفادة من الدوال – لزاماً عليك كتابة نفس الخطوات السابقة مرة أخرى .

وبدلاً من ذلك تستطيع جعل خطوات حساب المتوسط عامة وتستخدمها وقتما شئت ، عـن طريـق كتابة الدالة ثم استدعائها وقتما تريد لتنفيذ المطلوب بدلاً من إعادة كتابتها.

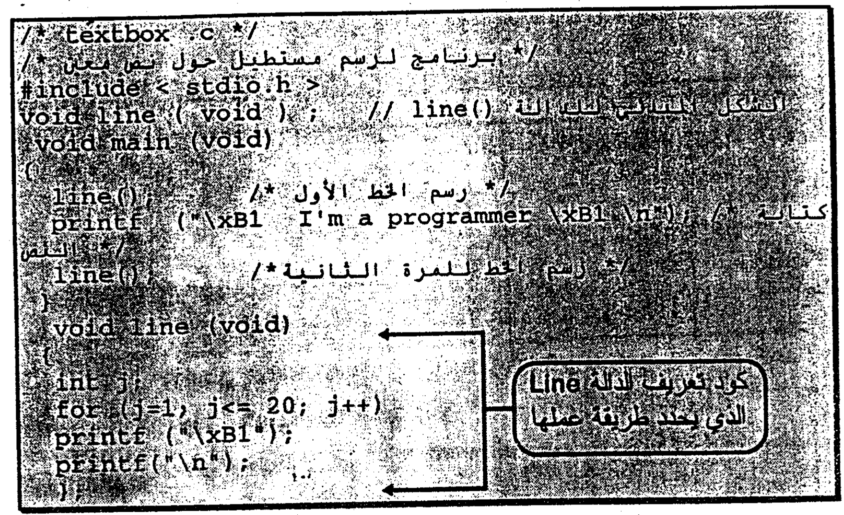
والدوال في لغة C إما دوال موجودة ضمن مكتباتها مثل دالة Printf ( ) وغيرها أو دوال يتم كتابتها بواسطة المستخدم ، لأداء مهمة متكررة.



2- **تنظيم كتابة وتقسيم البرنامج** . فمع تزايد خبرات المبرمجين في كتابة برامج الحاسب تأكد لديهم أن عمل برامج فرعية يتم استدعاءها عند الحاجة يساعد على تنظيم كتابة البرنامج ، كما انه يساعد على تقسيم العمل إلى وحدات ثم تجميعه مرة أخرى في البرنامج الكامل ، وهو ما يمثل الحالة المثالية في تصميم النظم الكبيرة التي تحتاج إلى أكثر من مبرمج للعمل فيها . مما يجعل البرنامج أسهل وأقرب إلى الفهم عند محاولة تعديله.

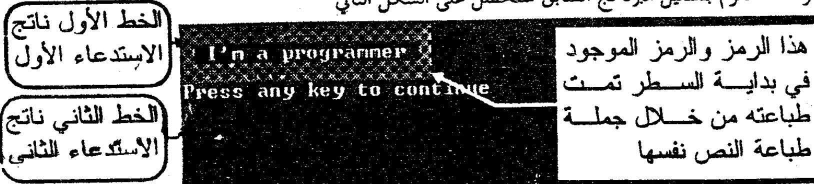
ويمكنك التفكير في الدالة على أنها شخص تستدعيه لتنفيذ مهمة ما ، فمثلاً لو أنك تطلب من أحد موظفيك إعداد تقريراً عن الميزانية وتتلقى منه هذا التقرير كل شهر ، فإنك في البداية تخبره كيف يعد هذا التقرير ثم يكفيك بعد ذلك أن ترفع سماعة التليفون لتقول له في نهاية كل شهر " الميزانية " وهو سيؤدي المهمة كاملة كما علمته.

بالضبط هذا هو عمل الدالة تقوم بتصميمها وتحدد المهمة التي يجب أن تؤديها وتعطيها اسماً ثم تطلب تنفيذها في أي وقت والمثال التالي يوضح ذلك .

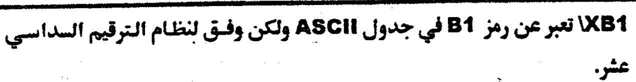


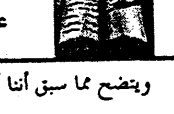
```
/* textbox .c */
/* برنامج لرسم مستطيل حول نص يعين
الشكل التالي الذي يلية */
#include < stdio.h >
void line ( void ) ;   // line() 
void main (void)
{
 line();        /* رسم الخط الأول */
 printf ("\xB1  I'm a programmer \xB1 \n"); /* كتابة النص */
 line();        /* رسم الخط للمرة الثانية */
}
void line (void)
{
 int j;
 for (j=1; j<= 20; j++)
 printf ("\xB1");
 printf("\n");
 }
```


وعندما نقوم بتشغيل البرنامج السابق ستحصل على الشكل التالي



والذي يُظهر النص محاطاً بالمستطيل المكون بواسطة رمز ASCII المسمى B1 .





XB1\ تعبر عن رمز B1 في جدول ASCII ولكن وفق لنظام الترقيم السداسي عشر.

ويتضح مما سبق أننا أشرنا إلى الدالة في البداية قبل دالة main() للإعلان عنها قبل استغلالها في الدالة الرئيسية التي تمثل البرنامج الذي سيستدعي الدالة ثم عرفنا الدالة في نهاية البرنامج (يمكن كتابتها في أي موضع من البرنامج) ، ومن البرنامج أيضاً استخدمنا الدالة مرتين لرسم خط قبل وبعد النص.

تركيب الدالة

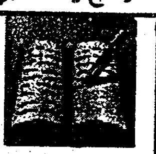
المثال السابق يمكن النظر إليه على أنه برنامجين

1 – البرنامج الكبير أو الأصلي الذي يدير العملية بالكامل ويتم من خلاله استدعاء البرنامج الفرعي الـــذي يرسم الخط في إحدى مراحله والذي يبدأ السطر.

```
Void main (void)
```

2 – البرنامج الفرعي الذي يتم استدعاؤه بمجرد ذكر الاسم الدال عليه والذي تم تعريفه (في موضع ما من البرنامج الرئيسي) وهو في هذا المثال الدالة Line() التي استدعيت مرتين.

وفي الحقيقة فإن كلاً منهما هو عبارة عن دالة Function فقد ذكرنا في الفصل الأول عند الحديـــث عـــن تركيب البرنامج في لغة C – أن Main() هي دالة ميزتها الأساسية هي أنها تنفـــذ أولاً قبـــل أي دالـــة في البرنامج ولذلك فهي التي تتلقى أولوية التحكم من نظام التشغيل بمجرد بدء تشغيل البرنامج



دالة main() يتم تنفيذها في بداية تشغيل البرنامج حتى ولو كانت مكتوبة بعد عدة دوال في ملف المصدر.

وفي المثال السابق قامت دالة main() باستدعاء دالة Line() لتنفيذ مهمتها وتم الاستدعاء مرتين لتنفيذ المهمة مرتين.

وباختصار هناك ثلاثة أمور يجب أن تلتفت إليها عند استخدام أي دالة

أ – تعريف الدالة Function Definition .

ب – استدعاء الدالة Function Call .

ج – الشكل المبدئي للدالة Function Prototype .

ودعنا نتناول كل منها بالتفصيل

أ - تعريف الدالة

ويطلق على السطور التي تحتوي كود الدالة نفسها وتحدد مهمتها وتمكنها من أداء وظيفتها وتبـــدأ أساســـاً بتعريف اسم الدالة كما سبق

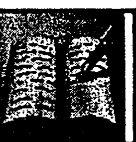
```
void line (void)
{
------------
```

ولاحظ أننا لم نضع (;) في نهاية السطر لأنه سطر غير منتهى .

ويعتبر هذا السطر هو سطر الإعلان عن الدالة Declaration .

ومن خلاله استخدمنا كلمة void الأولى لتعريف المترجم أن الدالة line() لـــن تعيـــد قيمـــة value إلى البرنامج الرئيسي الذي استدعاها أي أنها ستنفذ مهمة منتهية وفقط .

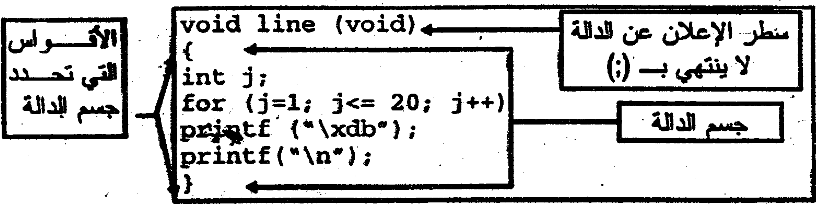
كلمة void الثانية بين الأقواس تعني أن الدالة لا تحتاج إلى وسيطات .



الحالة العامة للدالة أنها تأخذ قيمة أو أكثر كوسيطات وأنها تعيد إلى البرنامج الذي استدعاها نتيجة عمليات الحساب التي أدتها وسيلي مناقشة مثل هذه النوعيات.

وهذا السطر – سطر الإعلان – لا يؤدي أي مهمة سوى أخبار المترجم بأن تعريف دالة ما قد بدأ ولذلك فهو لا ينتهي بـــ (;) كباقي اسطر لغة C .

ويلي هذا السطر جسم الدالة الذي يتم من خلاله تحديد مهمتها ، وهو يكون محصوراً بين أقواس { } .



ب - استدعاء الدالة Calling the Function

يتم استدعاء الدوال في لغة C من خلال البرنامج الرئيسي أو دالة main() بذكر اسم الدالة سواء كـــانت الدالة المستدعاة من دوال لغة C الموجودة ضمن مكتباتها أو دالة من ابتكار المبرمج .



لو كانت الدالة المستدعاة من دوال المكتبات فيجب أن تضيف في بداية البرنامج الملف الذي يحتويها من خلال كلمة # include .

ولكن يجب ملاحظة أن كتابة الأقواس ( ) بعد اسم الدالة هي التي تلفت نظر المترجم – Compiler أنك تقصد بهذا الاسم اسم دالة ، فيبدأ في البحث عنها وعند الاستدعاء فإن جملة الاستدعاء هذه تكـــون جملـــة منتهية ولذلك فهي تنتهي بـــ (;) مثل

```
line();
```

وهو ما ينتج عنه نقل التحكم إلى جسم دالة ()line حتى يتم الانتهاء ثم العـودة مـرة أخـرى إلى دالـة .main()

ج -. الشكل المبدئي Prototype

وهو العنصر الثالث في التعامل مع الدوال التي يكتبها المبرمج بنفسه ، وهو عبارة عن السطر الـذي يسبق الدخول إلى دالة ()main مثل.

```
void line(void) ;  /*  الشكل المبدئي للدالة  */
```

ويحدد اسم الدالة ونوعيات البيانات المستخدمة فيها :

ولاحظ أن السطر ينتهي بـ (;) على عكس سطر تعريف أو إعلان الدالة ، وهو الفارق الوحيد في الكتابة بينهما ، ولكن ما فائدة هذا السطر؟؟!

فائدته أن يتم من خلاله تعريف المترجم أن البرنامج يحتوي على دالة وأنها سوف تعيد إلى البرنامج الرئيسي قيمة بنوع بيانات معين وأنها ستستخدم وسيطات Arguments عددها كذا ... ونوع بياناتها كذا ...

لن تعيد قيمة ← void line (void) → لن تأخذ وسيطات

ويجب أن يسبق هذا السطر أي استدعاء للدالة داخل البرنامج وإلا فلن يتعرف المترجم عليها.

لابد للدالة من ثلاثة عناصر

1. **سطر الشكل المبدئي وهو يسبق دالة main لتعريف المترجم بوجـود الدالة في الملف وتحديد أنواع البيانات المستخدمة معها.**
2. **سطر الإعلان عن الدالة وهـو أول سطر في تعريف الدالـة . ويخبر المترجم بدء عملية تعريف الدالة وأن السطور التالية لـه هـي جسم الدالة الذي يحدد وظيفتها.**
3. **استدعاء الدالة ويتم في أي موضع بذكر أسمها متبوعـاً بالأقواس ( ) التي قد تحتوي على وسيطاتها إن كانت لها وسيطات.**

تعليق على المتغير المحلي

استخدمنا في المثال السابق المتغير J الذي تم الإعلان عنه من داخل الدالة line() ولذلك فهو متغير معروف فقط للدالة ولا يمثل أي قيمة في البرنامج الرئيسي main() فمثلاً لو كتبنا الجملة التالية

```
printf ("%d",J);
```

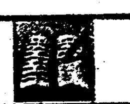
في البرنامج الرئيسي أي خارج دالة Line( ) ستحصل على خطأ ترجمة Compiling error يعتبر استخدام متغير لم يتم الإعلان عنه لأن البرنامج الرئيسي لا يعرفه.

كذلك لو أننا أعلنا عن متغير آخر باسم J في الدالة main() سيكون منفصل تماماً عن المتغير J المحلي في الدالة line() .

هذه نقطة هامة جداً في مجال تعريف وإعلان المتغيرات وتسمى إمكانية التعرف على المتغير من عدمه بواسطة الدالة الرؤية Visibility فكل دالة ترى المتغيرات المحلية التي تم إعلانها بداخلها لكنها لا تعرف أي متغيرات معلن عنها في دوال أخرى .. حتى لو تم الإعلان عنها داخل دالة main() نفسها والوسيلة الوحيدة لتعريف الدالة بمتغير غير معلن عنه داخلها هي الإعلان عنه في صورة متغير عام global .



رؤية المتغير variable Visibility هي إمكانية التعرف عليه داخل الدالة وقد يطلق على المتغير المحلي كالذي استخدم في المثال السابق متغير تلقائي automatic لأنه يتم إيجاده ثم تدميره في نهاية الدالة تلقائيا . وتسمى الفترة التي عاشها المتغير في ذاكرة الحاسب " منذ الإعلان عنه حتى نهاية الدالة " تسمى عمر المتغير life time .



سيأتي حديث تفصيلي عن عمر المتغير ورؤيته في الفصول التالية عند الحديث عن طرق التخزين storage Types .

الدوال التي تعيد قيمة

تعاملت حتى الآن مع دوال كلها تؤدي مهمتها دون أن تُعيد قيمة أو ناتج عملياتها الحسابية إلى البرنامج الرئيسي ، وهذه هي الحالة الأكثر استخداماً كما لا يخفى عليك ، ولكي نستطيع تتبع هذه الحالة دعنا نناقش المثال التالي:

```
/* getlow.c */
/* test getlc function */
#include < stdio.h >
#include < conio.h >
char getlc(void);     /* الشكل المبدئي */
void main (void)
{
char chlc;      /* char returned */
printf("type   'a'   for   first   selection,   'b'   for
second:");
chlc = getlc( ) ;   /* get converted character */
switch(chlc) /* print msg, depending on char */
{
case 'a':
printf("\nyou typed an 'a' ");
break;
case 'b':
printf("\nyou typed a 'b' ");
break;
default:
printf("\n you chose a non-existent selection ");
}
}
/* getlc */
/* returns character */
/* converts to lowercase if in uppercase */
char getlc (void)
{
char ch; /* char from keyboard */
ch = getche ( ); /* get character */
if (ch > 64 && ch < 91 ) /* if it is uppercase */
ch = ch + 32 ; /* add 32 to convert to lower */
return (ch); /* return charater to caller */
}
```

ويتضح من الكود السابق أننا أعلنا في السطر

```
char getlc (void);
```

وهو إعلان عن الشكل المبدئي لدالة getlc() لا تأخذ وسيطات في حين أنها تعيد قيمة عبارة عن حـــرف char .

وتتلخص مهمة البرنامج في تحويل الحروف الصغيرة Small إلى كبيرة والعكس ويأتي استدعاء الدالة مـــن خلال التعبير.

```
Chlc = getlc ( ) ;
```

الذي يتلقى ناتج الدالة في المتغير chlc الذي تم إعلان عنه في بداية البرنامج ويأتي تعريف الدالـــة في نهايـــة الكود، حيث يبدأ بالسطر

```
Char getlc (void)
{
```

ثم تتوالى الأسطر حيث تعلن عن متغير حرفي ch ، ونستخدم إحدى دوال المكتبات getche( ) والتي تعمل على قراءة حرف من لوحة المفاتيح وإظهاره على الشاشة ، وهي لا تنتظر ضغط مفتاح له .

بعدها تختبر الدالة موضع الحرف في جدول ASCII حيث تكون الأحرف الكبيرة محصورة بين الأرقام 64 حتى 91 وتم الاختبار بالجملة

```
if (ch > 64 && ch < 91)
```

فإن تحقق الشرط يتم تنفيذ السطر.

```
ch = ch + 32 ;
```

لتحويل الحرف إلى صغير

ثم نستخدم جملة Return لإعادة هذه القيمة إلى البرنامج الرئيسي.

جملة Return

في مثال textbox.c رأينا أن الدالة عندما تنتهي ويصل التنفيذ إلى القوس { يعود التحكم مرة أخرى تلقائياً إلى البرنامج الذي استدعى الدالة ، لكن ما هو الحال إذا لزم الأمر أن يتم ذلك قبل نهاية الدالة أو بشكـــل متعمد لظروف عمل البرنامج ؟ !!

يتم ذلك عن طريق الجملة Return() بحيث نكتب القيمة المراد إرجاعها إلى البرنامج الأصلي بين الأقواس، وبذلك فهي ذات فائدتين.

a- العودة الفورية من البرنامج الفرعي إلى البرنامج الرئيسي.

b- إعادة القيمة بين الأقواس إلى البرنامج الرئيسي.

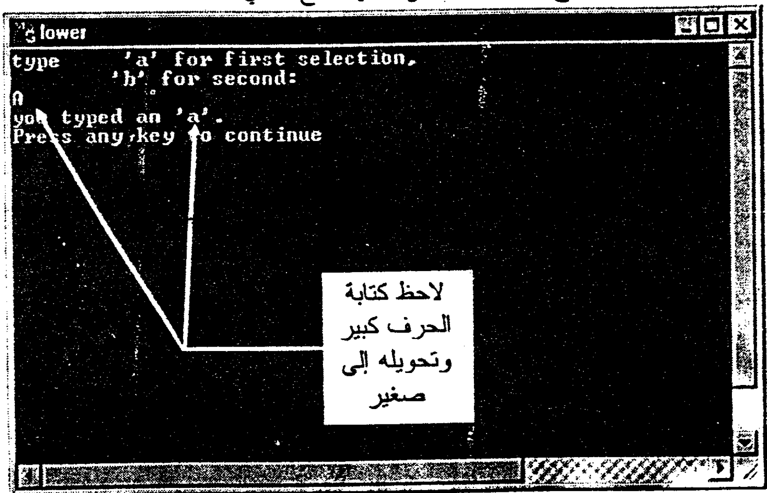


- **لا يشترط وجود جملة return() في نهاية الدالة.**
- **قد تكتب return بدون الأقواس إذا كانت لـن تعيـد قيمـة إلى البرنـامج الرئيسي.**

وعلى ذلك يمكننا كتابة المثال السابق على الصورة التالية:

```
/* modified get letter */
/* returns character */
/* converts to lower case if needed */
char getlc (void)
{
char ch;
ch = getche ();      /* gets character from keyboard*/
if (ch>64 && ch <91)     /* اختبار أن الحرف كبير */
return (ch+32);      /* للتحويل إلى حرف صغير */
else
return(ch);
}
```

وبالتالي عند تشغيل البرنامج السابق بعد التعديل سيكون الناتج كالتالي :



حدود Return()

في إنهاء الحديث في هذه النقطة يجب أن نذكر هنا أن Return() محدودة بإعادة قيمة واحدة من الدالة إلى البرنامج الرئيسي ، ولا يمكنها إعادة أكثر من قيمة ، فإن كانت هناك حاجة إلى ذلك يجب استخدام أسلوب مختلف عما سبق وسنناقشه في الفصول التالية لأنه يحتاج إلى معرفة بالمؤشرات Pointers وعناوين الذاكرة.



جملة Return تستطيع أعاده قيمة واحدة فقط من الدالة إلى البرنـــامج الرئيسي.

استخدام الوسيطات

الدوال التي تعاملنا معها حتى الآن دوال ذات وظائف محددة جامدة إلى حد ما بمعنى أنـــها لا تحتمل أي تغير في طريقة تنفيذها فبمجرد استدعاء الدالة تؤدي ما هي مصممة عليه دون أي تدخل خارجي الحالة العامة لاستخدام الدوال تختلف عن هذه الطريقة .

فنحن في الغالب نحتاج إلى جعل الدالة في صورة عامة وتغير طريقة تنفيذها على حسب موضع استدعائها.

فمثلاً دالة تحسب المتوسط الحسابي قد تغيير عدد المدخلات لها، فمرة تحسب متوسط خمسة أرقام ، وأخرى قد تحتاج إلى حساب متوسط عشرون رقماً ... وبذلك لابد من وضع قدر من المرونة في الدالـــة لتصلـــح للاستخدام في المواقف المختلفة. -

والوسيلة الوحيدة للتحكم في الدالة وطريقة تغيرها هي الوسيطات Agreements ، وهي القيم التي يتـــم وضعها للدالة بين الأقواس لتنفيذ مهمتها عليها.

وبالفعل نحن قد استخدمنا الوسيطات من قبل مع دوال مثل Printf() حين كتبت

```
Printf("  I  will be Professional C programmer .... Soon
")
```

فالعبارة الموجودة بين القوسين ( ) هي وسيطات الدالة Printf() وإليك المثال التالي الذي يعطيك فكرة عن كيفية إدخال وسيطات إلى دالة مكتوبة بواسطة المبرمج.

```
/* Scorew.cpp*/
/* a program to draw demonstration to a game score */
#include < stdio.h >
void bar (int) ;          /* الشكل المبدئي */
void main (void)
{
      int cars ;
int houses;
int trucks;
int earoplanes ;
// القيم التالية للمتغيرات يتم الحصول عليها من برنامج فرعي
cars = 15;
```

```
houses = 9 ;
trucks = 15 ;
earoplanes = 8 ;

printf ("\tcars\t\t") ;        /* طباعة كلمة cars */
bar (cars) ;          /* رسم خط بياني يوضح عدد
الوحدات */
printf ("\thouses\t\t") ;
bar (houses ) ;
printf ("\ttrucks\t\t") ;
bar (trucks) ;
printf ("\tearoplanes\t") ;
bar (earoplanes) ;
}
/* bar ( ) */
/* a function to draw a live to represent the score of
a game */
void bar (int score )
{
int I ;
for (I=1; I<= score ; I++ )
printf ("=");
printf ("\n");
}
```

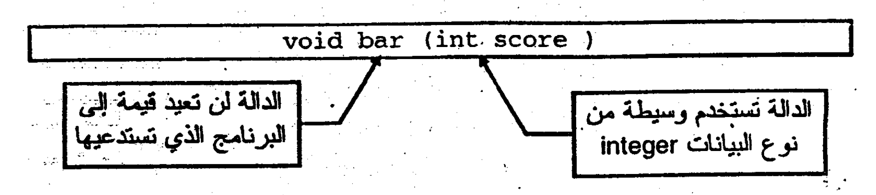
تعريف دالة ( )bar التي ترسم الخط البياني الأفقي

المثال السابق يستخدم في رسم بياني يظهر في نهاية إحدى الألعاب لعرض نتائج اللعبة حيث يتم أثناء اللعب محاولة تحطيم وحدات مختلفة من السيارات والمنازل والشاحنات والطائرات بواسطة طائرة يتحكم فيها اللاعب... والبرنامج لا يدخل في تفاصيل اللعبة لكنه يعمل بعد انتهائها لعرض نتائج اللعبة في صورة بيانية.
ففي بداية البرنامج كان متاحاً القيم Airplanes , Trucks , Houses , Cars التي يمثل كل منها رقم صحيح Integer .

والمطلوب هو وضع دالة تأخذ في كل مرة قيمة من هذه القيم وترسم على أساسها خطاً أفقياً يبين كمية ما حققه اللاعب ، هذه الدالة هي دالة ( )bar التي تم تعريفها في نهاية الملف السابق وهي تستدعي في بدايته بعد كتابة الكلمة المعبرة عن القيمة ثم يبدأ تنفيذ الدالة.

وبالطبع فإن المتغيرات التي توضع في بداية البرنامج وتحدد القيم المختلفة تكون ناتجة من دوال أخرى تم تنفيذها من قبل لحساب أو عد ما يحققه اللاعب ؟أثناء تشغيل اللعبة.

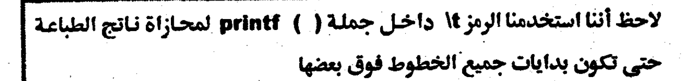
وما يعنينا هنا أن الدالة ( )bar تم استخدامها عدة مرات لترسم خطاً أفقياً تحكمنا في طوله على أساس ما يحققه اللاعب بواسطة الوسيطة score من النوع integer كما تم الإعلان عنه في السطر



وعند تشغيل البرنامج ستحصل على النتيجة التالية :

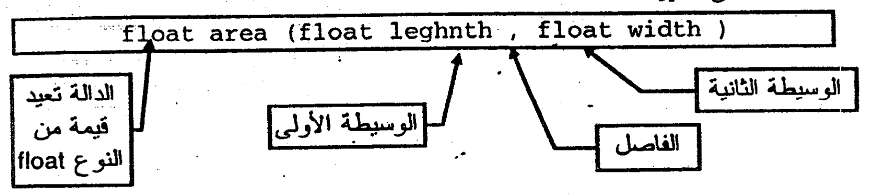
```
score
            cars                ===============
            houses              =========
            trucks              ===============
            earoplanes          ========
Press any key to continue_
```





لاحظ أننا استخدمنا الرمز t\ داخل جملة ( ) printf لمحازاة ناتج الطباعة حتى تكون بدايات جميع الخطوط فوق بعضها

وتستطيع إعطاء أكثر من وسيطة إذا كانت الدالة تحتاج إلى ذلك فمثلاً لو أن الدالة تحسب مساحة مستطيل فإن الوسيطات المفروض توفيرها يجب أن تكون الطول والعرض، ويمكن تحديد ذلك في سطر الإعلان عن الدالة بجعله على الصورة



والجدير بالذكر في هذه النقطة هي أن وسيطات الدالة قد تكون

- رقم ثابت مثل bar(50)
- متغير مثل bar(cars)
- تعبير رياضي مثل bar(cars+houses)

ولنتذكر دائماً أن أي متغير يتم إعطاؤه كوسيطة للدالة - تسمى هذه العملية تمرير المتغير للدالة - لا يتم إعطاؤه بنفسه لكن تعطي الدالة نسخة من المتغير بحيث أن الدالة لا تستطيع تغيير المتغير الأصلي ، بمعنى لو أن لديك متغير يسمى count أعطيته كوسيطة للدالة f() وكان من بين خطوات الدالة ما يغير قيمة count إلى قيمة أخرى فإن ذلك لا يؤثر على المتغير الأصلي ولكن أي قيم داخل الدالة يأخذها count تكون مؤقتة للحساب داخل الدالة ولا تؤثر على المتغير الأصلي.

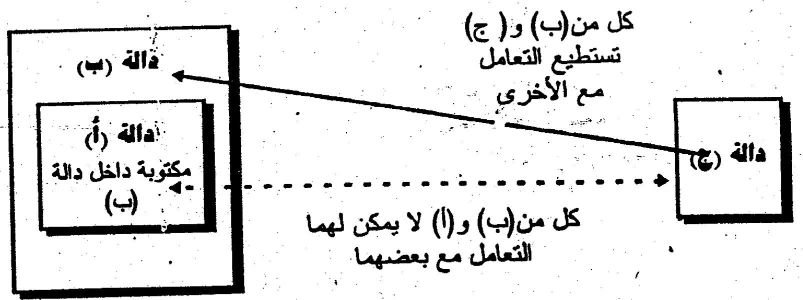
الوسيلة الوحيدة لتغيير قيمة **count** في هذه الحالة هي من خلال مؤشر يعدل قيمته في عنوانه من الذاكرة ، وسيلي شرح هذه النقطة في فصل المؤشرات.

استخدام أكثر من دالة

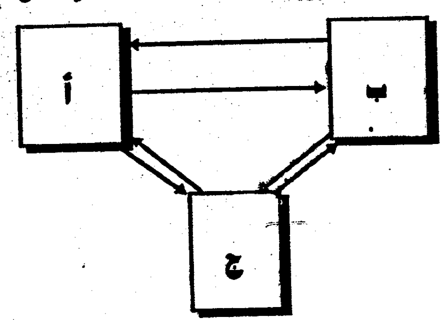
تستطيع أن تستخدم أي عدد من الدوال في برنامج لغة C ، وتستطيع أي دالة داخل البرنامج أن تستدعي أي دالة أخرى للتنفيذ من داخلها.

ويجب أن ننبه هنا إلى خاصية هامة لبرامج لغة C في التعامل مع الدوال يختلف عن اللغات الأخرى التي تستخدم الدوال أو الإجراءات ففي لغة مثل باسكال على فرض أن لدينا دالة (أ) تم الإعلان عنها من داخل دالة أخرى (ب) فإن كان هناك دالة ثالثة (ج) في البرنامج فإنها لا تستطيع رؤية الدالة (أ) على أساس أن (أ) تستطيع القول أنها دالة داخلية في (ب).

الوضع في لغة C يختلف فأي دالة مكتوبة في أي موضع من البرنامج تكون مرئية لجميع الدوال ، أي أنك لا تستطيع إنشاء دوال داخلية حتى لو حاولت والشكل التالي يوضح طريقة التعامل بين الدوال في لغة C واللغات الأخرى .

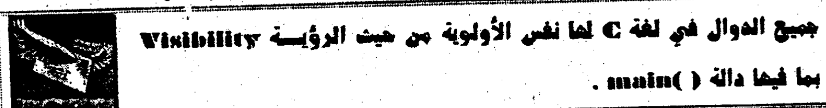


في لغة مثل باسكال لا يمكن لدالة (ج) الوصول للدوال الداخلية مثل دالة (أ).



وفي لغة C يمكن لجميع الدوال رؤية بعضها البعض فهي لا تسمح بالدوال الداخلية.

و C في هذه النقطة تشبه لغة Basic حيث لا يمكنك جعل إجراء فرعي غير مرئي لباقي الإجراءات.



جميع الدوال في لغة C لها نفس الأولوية من حيث الرؤيــة Visibility بما فيها دالة main( ) .

المتغير الخارجي External Variable

حتى الآن كانت جميع المتغيرات التي تعاملنا معها متغيرات يتم الإعلان عنها داخل الدوال وبالتالي فإنها تكون مرئية داخل الدالة ولا يمكن استخدامها خارجها وهو ما أسميناه متغير داخلي أو محلي أو تلقائي ، وبـــالرغم من أن استخدام المتغيرات المحلية مفضل كما سبق ذكره لكن هناك بعض الحالات الـــتي نحتـــاج فيـــها إلى استخدام متغيرات مرئية لجميع الدوال وليس دالة واحدة.

وفي مثل هذه الحالة نستخدم متغير خارجي أو عام " كوني " global ، وإليك المثال التالي الذي نســتخدم فيه هذه النوعية من المتغيرات.

```
/* مثال على استخدام المتغيرات الكونية مع عدة دوال */

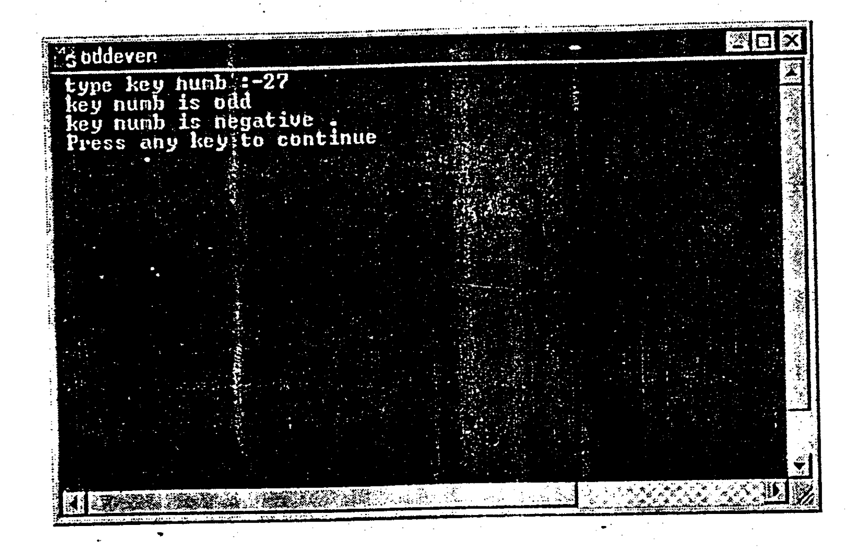
# include <stdio.h >

void oddeven (void) ;        // functions prototypes
void negative (void) ; // الإعلان عن المتغير
signed int key_numb ; // maina() بداية دالة
void main (void)
{
printf (" type key numb :");
scanf ("%d" , &key_numb ) ;
oddeven ( ) ;                /* استدعاء دالة */
negative ( ) ;               /* استدعاء دالة */
}
// odd even() تعريف الدالة
/* cheks if key_numb is odd or even */
void oddeven (void)
{
if ( key_numb %2 )
printf ("key numb is odd \n") ;
else
printf (" key numb is even . \n");
}
// negative() تعريف الدالة
/* checks if keynumb is negative */
void negative (void)
{
if ( key_numb < 0)
printf (" key numb is negative .\n ");
else
printf (" key numb is postive . \n");
}
```

في المثال السابق استخدمنا دالتين ( )Oddeven التي تختبر هل الرقم زوجي أو فردي والدالة ( ) negative التي تختبر إشارة الرقم بالإضافة إلى الدالة الرئيسية ( ) main .

وكلها أمكنها الوصول إلى متغير واحد يقوم المستخدم بكتابته في بداية تشغيل البرنامج ولكي يتم ذلك أعلن عن المتغير على أنه external or global بوضعه في بداية البرنامج حتى قبل الدخول إلى دالة ( ) main .

وعند تشغيل البرنامج ستحصل على الشكل التالي :





- المتغير الخارجي يتم الإعلان عنه خارج أي دالة وحتى خارج دالة main ( )
- لا تتعجل في استخدام المتغيرات الخارجية إلا إذا كنت فعلا مضطر إلى ذلك حتى لا نفقد كم من الذاكرة قد يؤثر على سرعة الحاسب وكفاءة تشغيل البرنامج.

استخدام Arrays كمدخلات إلى Functions:

سيتم من خلال مثال (١٣-١) عرض كيفية إدخال array من نوع integer إلى function. فعند شرح arrays في الفصل التاسع "مقدمة إلى مفهوم تراكيب Arrays في لغة ++C" تم وصف كيفية قيام متغير array بتحديد عنوان أساسي في الذاكرة. وتتم هذه العملية بشكل غير مرئي للمبرمج. لذلك فكل ما يجب عمله هو إدخال array إلى function، ثم معالجة المتغيرات بالطريقة العادية الخاصة بال arrays.

مثال (١٢-١): كيفية استخدام Array في عمليات الإدخال

```
#include <iostream.h>

#define MAX 10
```

```
void Show(int n[MAX]);

main()
{
// numbers can store 10 integers.
   int numbers[MAX] ={12,99,77,34,6,45,199,38,123,91};

   Show(numbers);
   return(0);
}

v[illegible]v(int n[MAX])
{
   int index;

   for (index = 0; index < MAX; index++)
   {
      cout << "ARRAY ITEM  " << index
         << " IS " << n[index] << endl;
   }
}
```

وتظهر ( ) Show هنا بشكل مماثل تمامًا لما سبق ذكره عنها، فهى لا تقدم نتيجة وتسمح بإدخال array من نوع integer إليها. ويجب هنا ملاحظة أنه يتم تعريف array، وليس متغير بسيط، باستخدام n[MAX]. فيتم إدخال الأرقام إلى function، حيث يتم نسخها فى المتغير الداخلى n. وبداخل function، يتم كتابة كل القيم الموجودة فى array على الشاشة باستخدام for loop. وفيما يلى الشكل الذى ستظهر به الشاشة:

```
ARRAY ITEM 0 IS 12
ARRAY ITEM 1 IS 99
ARRAY ITEM 2 IS 77
ARRAY ITEM 3 IS 34
ARRAY ITEM 4 IS 6
ARRAY ITE[illegible]
ARRAY IT[illegible]
ARRAY IT[illegible] 38
ARRAY ITEM 8 IS 123
ARRAY ITEM 0 IS 91
```

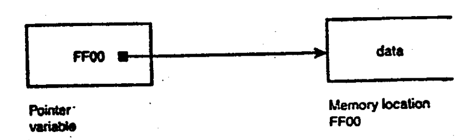
نظرة عامة على مفهوم Pointers:

لقد تم حتى الآن تجنب موضوع كود pointers، والمسمى readed p
word. وهناك سببان لذلك. أولاً: إن هذا الموضوع يسبب قلقاً لمعظم المستخدمين. ثانياً: إن لا يبدو بالأمر الصعب بعد أن يتفهمه الدارس جيداً.

لقد تم فى الفصل الثانى "أهم مكونات لغة ++C" تقديم مفهوم المتغير. وهو عبارة عن صندوق يحتوى على نوع من أنواع البيانات مثل integer أو float. ومع ذلك، فأحياناً يفضل المبرمجون المتخصصون استخدام pointers (أو pointer variable، كما يسمونه) عندما يقوموا بالتعامل مع برامج تتطلب حيزاً كبيراً فى الذاكرة مثل تطبيقات قواعد البيانات أو المعالجة المباشرة للذاكرة التى تتم عند كتابة نظم التشغيل.

ويتمثل تعريف pointer فيما يلى: إن pointer variable (متغير pointer) هو متغير يحتوى على عنوان موقع تخزين البيانات فى الذاكرة.

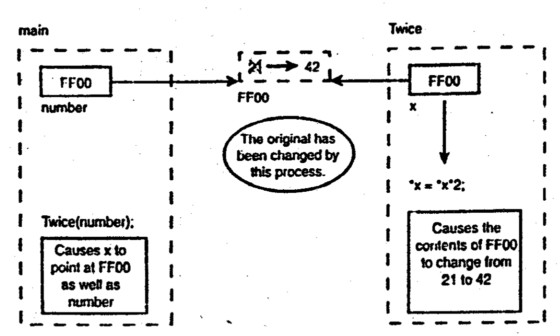
وبما أن هذا يبدو معقداً إلى حد ما، فيمكن توضيح الأمر من خلال هذا الشكل:



يشير متغير pointer إلى موقع تخزين البيانات فى الذاكرة.

تعريف مفهوم Pass by Reference:

يتلخص مفهوم pass by reference فى أنه عند إدخال متغير pointer الخارجى إلى function، فإن محتويات هذا المتغير، التى تتمثل فى العنوان، يتم نسخها فى متغير pointer الداخلى الذى سبق تحديده فى function header. ويعنى هذا أن هذين المتغيرين الداخلى والخارجى يشيران لنفس المكان فى الذاكرة لأن العنوان فى كلا المتغيرين يشير إلى موقع تخزين البيانات فى الذاكرة داخل الكمبيوتر. وبذلك يمكن معالجة ذلك الموقع المشترك فى الذاكرة من داخل function أو من خلال البرنامج الرئيسى لأن كلاهما يشتمل على تلك المتغيرات التى تشير إلى نفس هذا الموقع الذى يطلق عليه اسم shared memory (الذاكرة المشتركة)، وإليك الشكل التالى الذى يوضح هذا المفهوم:



شكل يوضح تسلسل الخطوات فى تنفيذ تقنية pass by reference.

أهم النقاط التى تعرض لها الشكل السابق:

لم يقتصر البرنامج فى مثال (١٣-٢) على توضيح مفهوم pass by reference، بل ألقى الضوء أيضاً على استخدامات pointers .

- يتم إنشاء المتغير pointer باستخدام السطر التالى. مع ملاحظة كيفية استخدام النجمة (*) للتمييز بين متغير pointer والمتغير العادى.

```
int* number;
```

- يتم إنشاء وتكوين متغير عادى.

```
int Num1 = 77;
```

- والآن يجب نسخ محتويات المتغير العادى داخل منطقة تخزين البيانات فى الذاكرة التى يشير إليها متغير pointer. مع ملاحظة استخدام علامة &.

```
number = &Num1;
```

- يتم عرض محتويات منطقة تخزين البيانات فى الذاكرة التى يشير إليها الرقم باستخدام السطر التالى. مع ملاحظة استخدام النجمة (*) قبل هذا الرقم.

```
cout << *number << endl;
```

- والآن يجب إدخال العنوان الخاص بالبيانات إلى ( ) Twice، حيث تتضاعف محتويات تلك المنطقة الموجودة فى الذاكرة.

```
Twice(number);
```

- عند إخراج محتويات المتغير number مرة ثانية، ستكون محتويات هذا المتغير قد تضاعفت بالفعل. وبذلك تكون قد عالجت عنصر البيانات الأصلى.

```
cout << *number << endl;
```

عند تنفيذ خطوات هذا البرنامج، سيبدو نموذج المخرجات على الشاشة كمايلى :

```
77
154
```

مثال (١٢-٢): مثال يوضح مفهوم Pass by Reference

```
#include <iostream.h>

void Twice(int* x);

main()
{
   int* number;
   int Num1 = 77;

   number = &Num1;
   cout << *number << endl;
   Twice(number);
   cout << *number << endl;
   return(0);
}

void Twice(int* x)
{
   *x = *x * 2;
}
```

يجب توخى الحذر عند استخدام هذه الطريقة، لأنه يمكن أن تحدث آثار جانبية. ومع ذلك فالمبرمج ذو الخبرة سيجد ها طريقة مشجعة للدخول فى هذا المجال.

تحذير:

عند العمل فى نظام تشغيل حديث مثل برنامج Windows 95/98، يعتبر البحث فى الذاكرة أمراً شديداً الخطورة. فلكل نظام تشغيل أسلوبه الخاص فى إدارة الذاكرة حيث يتم تخصيص أماكن للموارد فى الذاكرة عند الحاجة إلى ذلك. وسيتم عرض هذا الموضوع مرة ثانية فى الفصل الثالث والعشرين "تخصيص الذاكرة" (Memory Allocation).

استعراض عملية بسيطة من عمليات ترتيب البيانات:

والآن بعد عرض مفاهيم معقدة مثل arrays و functions ، و pointers و pass by reference ، نحن بصدد العرض لبـ نامج يتم فيه ترتيب البيانات بشكل تصاعدى . ويتمثل الغرض الأساسى من هذا البرنامج فى تسجيل أربعة أرقام فى الذاكرة ، وتخزينهم فى شكل array ، ثم ترتيبهم ترتيباً تصاعدياً ، ليتم عرضهم بعد ذلك على الشاشة .

خطوات البرنامج:

يمكن وصف خطوات البرنامج على النحو التالى :

- يتم إنشـــاء array من نوع MAX intege، اسمها data ، ويتم بعد ذلك إدخال pointer يشير إلى العنصر الأول فيها إلى Get Data (باستخدام أسلوب pass by reference .
- وبذلك فإن الـ pointer الداخلى X يحمل الآن نفس عنوان تركيب array المسمى data، ولذلك يمكن لكلاهما استخدام نفس مكان تخزين البيانات فى الذاكرة (وهو المكان الذى يتم فيه تخزين البيانات المحفوظة فى عناصر array) .
- ثم يتم ببساطة تسجيل مجموعة من بيانات integers داخل GetData، ووضعها فى array .
- يجب توضيح فكرة أنه يتم استخدام أسلوب pass by reference ولن يكون هناك داع لتقديم أية نتيجة للبرنامج الرئيسى ، لأن الكتابة تتم مباشرة للمنطقة التى تشغلها array فى الذاكرة .
- يمكن عرض هذه البيانات على الشاشة باستخدام ( ) Show بعد تجميعها وتخزينها .



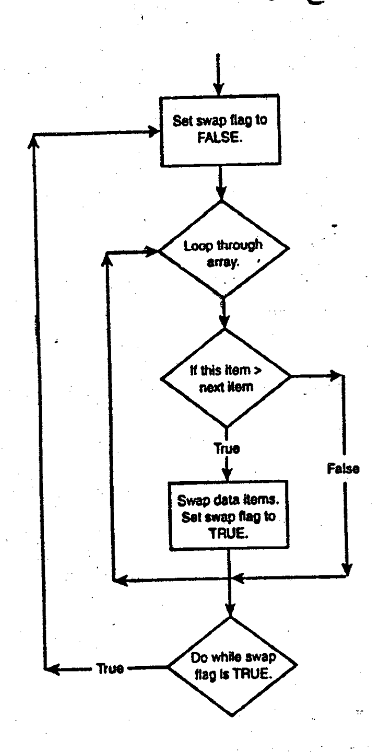
الآثار الجانبية

يستخدم مصطلح الآثار الجانبية فى لغة البرمجة فى الإشارة إلى النتيجة التى تحدث عندما يؤدى نشاط جزء من الكود إلى إحداث شئ آخر غير متوقع فى مكان آخر فى النظام. فعلى سبيل المثال، يمكن أن نلاحظ هذه النتيجة من خلال استخدام pass by reference عندما يشترك جزئين أو أكثر من الكود فى استخدام نفس موقع تخزين البيانات فى الذاكرة. وإذا تم تغيير بيانات هذا الموقع تحت تأثير استخدام كود معين، فإنه سيكون هناك مخاطرة فى تغيير ناتج نشاط هذا الجزء من الكود الذى يستخدم البيانات الموجودة فى هذا الموقع المشترك فى الذاكرة.

- والآن يمكن إدخال العنوان الخاص بالبيانات إلى( ) Sort وهى عبارة عن function عملية الترتيب، والتى تستخدم أسلوب bubble sort فى ذلك. وهو مفهوم معقد ويحتاج إلى مزيد من الشرح لأنه يتم استخدامه على نطاق واسع.

عرض لأسلوب Bubble Sort فى الترتيب:

يشرح الشكل التالى الخطوات المستخدمة فى عملية الترتيب، دون الإشارة لعمليات الإبدال التى ستحتاج إلى شرح أكثر.



شكل يوضح أسلوب bubble sort فى ترتيب البيانات

شرح عملية الإبدال:

يوضح الشكل التالى فى ثلاث خطوات كيفية إبدال محتويات العنصر a مع محتويات العنصر b . مع ملاحظة أن المتغير يشتمل على قيمة واحدة فقط فى كل مرة. بعد كتابة هذه القيمة ، فإنه لن يمكنه تذكر القيمة الأصلية .

a	b	c	
21	6		
21	6	21	1. Copy a to c.
6	6	21	2. Copy b to a.
6	21	21	3. Copy c to b.

خطوات إبدال محتويات العنصر a بمحتويات العنصر b.

تطبيق أسلوب Bubble Sort على البيانات من نوع integer:

والآن بعد فهم النظرية يمكن تنفيذ البرنامج الموجود فى مثال (١٣ - ٣)ويتم فيه إدخال أربعة integers بشكل عشوائى. وسيقوم البرنامج بتنفيذ الترتيب وإظهار النتائج. وسيكون نموذج المخرجات على هذا النحو الذى يمكن استبدال أرقامه بأى أرقام أخرى على شرط أن تكون من نوع integer :

```
Enter a number : 13
Enter a number : 4
Enter a number : 77
Enter a number : 42

BEFORE THE SORT
13
4
77
```

```
42

AFTER THE SORT
4
13
42
77
```

مثال (١٢-٢): إدخال البيانات إلى تركيب array معين بتحديد عنوان العنصر فى كل مرة

```
// Simple sort demo

#include <iostream.h>

#define MAX 4

void GetData(int x[MAX]);
void Sort(int x[MAX]);
void Show(int num[MAX]);

main()
{
   int data[MAX];

   GetData(data);
   cout << endl << "BEFORE THE SORT" << endl;
   Show(data);
   cout << endl;
   Sort(data);
   cout << "AFTER THE SORT" << endl;
   Show(data);
   cout << endl;
   return(0);
}

void GetData(int x[MAX])
{
   int loop;

   for (loop = 0; loop < MAX; loop++)
```

```
{
   cout << "Enter a number : ";
   cin >> x[loop];
  }
}

void Show(int x[MAX])
{
  int loop;

  for (loop = 0; loop < MAX; loop++)
  {
    cout << "Array item " << loop
<< " contains " << x[loop] << endl;
  }
}

void Sort(int x[MAX])
{
  int loop;
  int swapflag = 0;
  int temp;

  do
  {
    swapflag = 0;
    for (loop = 0; loop < MAX - 1; loop++)
    {
      if (x[loop] > x[loop+1])
      {
        temp = x[loop];
        x[loop] = x[loop+1];
        x[loop+1] = temp;
        swapflag = 1;
      }
    }
  }while (swapflag != 0);
}
```

تطبيق أسلوب Bubble Sort على البيانات من نوع string:

فى هذا الجزء، ستقوم بتجميع كل ما عرفته بالفعل لتطبيق أسلوب bubble sort فى الترتيب على array من نوع string. ويفيد هذا البرنامج كثيراً فى ترتيب الأسماء وفقاً للحرف الأبجدى.

استخدام Arrays متعددة الأبعاد:

بعد وصف كيفية معالجة arrays باستخدام pass by reference، سأستعرض الآن وصفاً مشابهاً مستخدماً array متعدد الأبعاد ويمكن أن تكون array ثنائية الأبعاد من نوع char، والتى هى فى الواقع عبارة عن array أحادية البعد من نوع string. ويقدم مثال (١٣-٤) البرنامج المسئول عن ترتيب قائمة من strings ترتيباً تصاعدياً وفقاً للترتيب الأبجدى. ومفهوم البرنامج هنا مطابق لمفهوم برنامج bubble sort السابق. ولكن أدى استخدام strings بدلاً من integers هنا إلى ضرورة استخدام string.h library.

حاول تشغيل البرنامج وقم بإدخال أربعة أسماء. وستبدو الأسماء الأربعة على الشاشة بالصورة التالية:

```
Enter a name : Crockett Davy
Enter a name : Wallace William
Enter a name : Ali Mohammed
Enter a name : Keegan Kevin

Crockett Davy
Wallace William
Ali Mohammed
Keegan Kevin

Ali Mohammed
Crockett Davy
Keegan Kevin
Wallace William
```

مثال (١٣-٤): استعراض Array من نوع String:

```
#include <iostream.h>
#include <string.h>

#define MAX 4
```

```
#define LENGTH 20

void GetData(char s[MAX][LENGTH]);
void Show(char s[MAX][LENGTH]);
void Sort(char s[MAX][LENGTH]);

main()
{
   char people[MAX][LENGTH];

   GetData(people);
   cout << endl;
   Show(people);
   cout << endl;
   Sort(people);
   Show(people);
   cout << endl;
   return(0);
}

void GetData(char s[MAX][LENGTH])
{
   int loop;

   for (loop = 0; loop < MAX; loop++)
   {
      cout << "Enter a name : ";
      cin.getline (s[loop],20);
   }
}

void Show(char s[MAX][LENGTH])
{
   int loop;

   for (loop = 0; loop < MAX; loop++)
   {
      cout << s[loop] << endl;
```

```
    }
}

void Sort(char x[MAX][LENGTH])
{
   int loop;
   int swapflag = 0;
   char temp[LENGTH];

   do

      swapflag = 0;
      for (loop = 0; loop < MAX - 1; loop++)
      {
         if (strcmp(x[loop],x[loop+1]) > 0)
         {
            strcpy(temp,x[loop]);
            strcpy(x[loop],x[loop+1]);
            strcpy(x[loop+1],temp);
            swapflag = 1;
         }
      }
   }while (swapflag != 0);
}
```

كيفية مقارنة Strings:

مما سبق يمكنك إدراك أن طريقة البرنامج هنا مطابقة لطريقة برنامج bubble sort ، وأن الاختلاف الوحيد يكمن فى أن array ثنائية الأبعاد لها عنوانين يجب تحديدهما . فالخطوات المتبعة فى تنفيذ functions هذا البرنامج هى نفسها خطوات التنفيذ فى البرنامج السابق فيما عدا أنه يتم استخدام هنا string.h library functions لتتلائم مع بيانات strings المستخدمة فى هذا البرنامج . وعلى الرغم من أن ( )Swap المسئولة عن عملية الإبدال تعمل بنفس الطريقة ، فإنه يجب فحص السطر الذى تبدأ به هذه الـ function .

```
if (strcmp(x[loop],x[loop+1]) < 0)
```

يدور هذا السطر حول ( ) strcmp، الموجودة فى string.h library، والتى تقوم بمقارنة الترتيب الأبجدى بين عنصر الإدخال الأول [loop] x وعنصر الإدخال الثانى x [loop+1]. وتكون نتيجتها إحدى القيم الثلاثة التالية التى تكون من نوع integer :

if x[loop] = x[loop+1] strcmp returns = 0

if x[loop] > x[loop+1] strcmp returns > 0

if x[loop] < x[loop+1] strcmp returns < 0

ويتم فحص قيمة integer الناتجة باستخدام جملة if لتحديد العمل المراد تنفيذه – والذى يتمثل هنا فى عملية الإبدال. فإذا كان العنصر الأول أكبر من العنصر الثانى، ستكون نتيجة ( ) strcmp قيمة أكبر من صفر. ولذلك تتم عملية الإبدال. وإذا كانت نتيجة ( ) strcmp قيمة تساوى صفر أو أقل من صفر، لن تتم عملية الإبدال.



المقارنة فى الإتجاه العكسى

بدلاً من استخدام ( ) strcmp للمقارنة بين العناصر تبعاً لأكبر من، يمكنك تغيير المقارنة إلى أصغر من:

```
if (strcmp(x[loop],x[loop+1]) > 0)
```

وبذلك تكون عكست اتجاه المقارنة.

أهم الدروس المستفادة:

- عند إدخال متغير من نوع array إلى function، فإن ما يحدث فى الواقع هو إدخال العنوان الموجود فى الذاكرة الذى يشير إلى موضع هذه البيانات.
- متغير pointer عبارة عن متغير يحتوى على عنوان موقع تخزين البيانات فى الذاكرة.

- عند إدخال متغير pointer خارجى إلى function ، فإن محتويات هذا المتغير ، التى تتمثل فى العنوان ، يتم نسخها فى متغير pointer داخلى سبق تحديده فى function header . وهذا يعنى أن هذين المتغيرين الداخلى والخارجى يشيران لنفس المكان فى الذاكرة لأن العنوان فى كلا المتغيرين يشير إلى موقع تخزين البيانات فى الذاكرة داخل الكمبيوتر . وبذلك يمكن معالجة ذلك الموقع المشترك فى الذاكرة من داخل function أو من خلال البرنامج الرئيسى لأن كلاهما يشتملان على تلك المتغيرات التى تشير إلى نفس هذا الموقع .
- bubble sort عبارة عن جزء من الكود- الذى يكون عادة عبارة عن function- يتم من خلالها التحقق مما إذا كانت البيانات مرتبة ترتيباً صحيحاً . وبالمثل ، يمكن استخدامها فى ترتيب integers ترتيباً تصاعدياً أو ترتيب الأسماء وفقاً للترتيب الأبجدى . ولا يعتبر أسلوب bubble sort الأسلوب الأكثر كفاءة فى الترتيب ، وإن كان يعد أبسط الأساليب وأيسرها من حيث الفهم .
- يمكن استخدام pass by reference كأداة لمعالجة متغير خارجى من داخل function ، لأن pass by reference تعطى function عنوان البيانات الخارجى . وبالتالى تؤثر أية معالجة تتم داخل function على البيانات الأصلية المخزنة فى الذاكرة الرئيسية .
- تستخدم ( ) strcmp ، المخزنة فى String.h library ، للمقارنة بين قيمتين من نوع string أو بين متغيرات من نوع string . فإذا كان الـ string الأول مطابق للـ string الثانى ، ستنتج قيمة تساوى صفراً . وإذا كان الـ string الأول يسبق string الثانى أبجديًا ، ستنتج قيمة سلبية أقل من صفر . وإذا كان الـ string الأول تال للـ string الثانى أبجديًا ، ستنتج قيمة إيجابية أكبر من صفر .



الفصل الثامن

المؤشرات والسجلات

الفصل الثامن
المؤشرات والسجلات

من خلال هذا الفصل سنتعامل مع موضوع من أهم العلامات المميزة للغة C وهو المؤشـــرات Pointers وهو موضوع قد ينظر إليه البعض ممن سبق لهم التعامل بلغة C على أنه صعب أو غير مهم.
وقد يكون هذا التصور نابع من كون مفهوم المؤشرات جديد عليهم لأنه لا مثيل له في لغات مثـــل Basic أو Pascal .
أو ربما طريقة استخدام الرموز Symbols لأداء هذه العملية تسبب بعض الخلط نتيجة لاســـتخدام الرمـــز الواحد في أكثر من غرض كما سنرى بعد قليل.
ما نود قوله في هذه النقطة هو أن موضوع المؤشرات ليس بهذه الصعوبة وكما اتفقنا في بداية الكتاب فـــإن أي مشكلة يمكن حلها بالمعرفة والتدريب ومن خلال هذا الفصل سنحاول التعرف على أهميـــة المؤشـــرات وطريقة عملها.

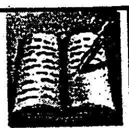
مفهوم المؤشرات

قبل الخوض في طريقة الاستفادة من المؤشرات دعنا نتعرف على طبيعة المؤشر فهو وسيلة للتعامل مع أحـــد المتغيرات أو المصفوفات دون الرجوع إلى المتغير مباشرة ويتم ذلك من خلال الاتصال بعنـــوان المتغـــير في الذاكرة variable address دون الإشارة المباشرة إلى المتغير باسمه.

ولكن لماذا نستخدم المؤشرات Pointers ؟

مادامت هناك طرق عادية للتعامل مع المتغيرات كما سبق ؟!!

أولاً – لأن المؤشرات تسمح للدالة بالتعديل في قيمة أحد المتغيرات في البرنامج الذي استدعى الدالة وهو ما لا يمكن تنفيذه سوى من خلال المؤشرات.



تذكر أننا عند الحديث عن تمرير المتغيرات للدوال ذكرنا أن الدالة تأخذ صورة من المتغير لإتمام حساباتها بواسطته ومهما تغيرت قيمة هذه الصورة داخل الدالة لا تأثير لها على المتغير الأصلي ، لذلك تعد الوسيلة الوحيدة لتغير قيمة مثل هذا المتغير هي من خلال مؤشر يعدل قيمته في الذاكرة.

ثانيا – بواسطتها يمكن إعادة أكثر من قيمة من الدالة لأن return لا تعيد سوى قيمة واحدة.

ثالثاً – تعد وسيلة مثالية في حالة تمرير مصفوفة أو نص من دالة لأخرى لأنها أسرع كثيراً.

وغيرها من الأسباب الأخرى التي ستتضح من خلال هذا الفصل.

تعريف المؤشرات Pointers

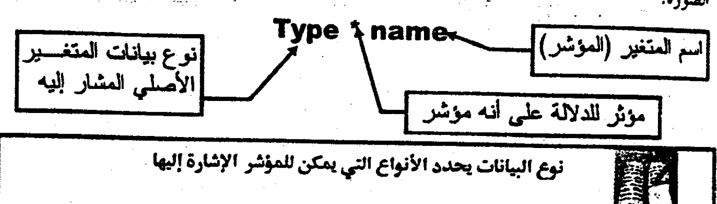
هو متغير يحفظ به عنوان في الذاكرة ذلك العنوان يحفظ به قيمة متغير آخر فعلى سبيل المثال قد يكون لدينا متغير x قيمته 100 وموجود في الذاكرة في العنوان 1000 فإن أنشاءنا مؤشر للمتغير x وليكن y فـــإن y يحفظ بها القيمة 1000 التي تمثل عنوان قيمة x وتكتب.

```
y = & x ;
```

y تساوي عنوان x .

الإعلان عن المؤشر Pointer Declaring

يتم الإعلان عن المتغيرات التي تمثل مؤشرات بنفس الطريقة التي يعلن بها عن المتغير العـــادي وهـــي علـــى الصورة.



- المتغير الذي يعمل كمؤشر يجب أن يعلن عنه مـن البداية علـى أنـه مؤشر.
- يتعرف المترجم على المتغير كمؤشر بواسطة الرمز * قبـل الاسم في جملة الإعلان مثل int *star; للإعلان عن متغـير مؤشـر star يستخدم للإشارة إلى أرقام صحيحة integer .

المؤثرات المستخدمة مع المؤشرات

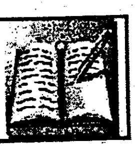
ذكرنا من قبل أنه هناك مؤثرين يتعاملان مع المؤشرات هما * , & ويمكنك التفكير في المؤثر & على أنــــه كلمة "عنوان" فلو كتبنا.

```
Y = &x,
```

يمكن قراءتها لا تساوي عنوان x .

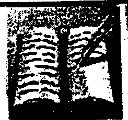
بينما المؤثر * يمكن قراءته " القيمة الموجودة في العنوان" ، وبالتالي فإن

```
x = *y ;
```



لا داع لإحداث أي خلط نتيجة كون مؤثر * يستخدم في عمليات الضرب ومؤثر & يستخدم كمؤثر منطقي.

ويجب أن تتأكد من أن المؤشرات التي تستخدمها تشير إلى أنوع البيانات الصحيحة فعلى سبيل المثال عندما تعلن عن مؤشر للإشارة إلى متغير من النوع integer لا تستخدمه في الإشارة إلى رقم عشري float ذلك لأن المؤشر في هذه الحالة سيأخذ القيمة الموجودة في 2بايت الأول من حيز الرقم العشري (8بـــايت) دون ظهور مشكلة في عمليات الترجمة Compiling.



- المتغير من النوع integer يمثل 2 بايت ، المتغير من النوع float يحتل 4بايت فيما عدا نظام windows95 فيكون فيه int يحتل 4 بايت.
- بعض المترجمات compilers قد يصدر تحذير في هذه الحالة وبعضها قد لا يشعر بالخطأ خاصة مترجمات ما قبل ++C .

تعبيرات المؤشرات Pointers Expression

بصفة عامة التعامل مع المؤشرات يتم كما يحدث مع أي متغير ... وبالتالي ينطبق على التعبيرات والجمـــل التي تستخدم المؤشرات كل ما ينطبق على المتغيرات العادية لكن هناك بعض الاعتبـــارات الخاصـــة الـــتي سنناقشها فيما يلي:

1 - تخصيص القيم لمؤشرات

كما يحدث مع جميع المتغيرات فإن استخدام المؤثر = يؤدي إلى وضع القيمة الموجودة على يمينه في المتغير الموجود على يساره.

وعلى ذلك قد يتم وضع مؤشر في الجهة اليمين لوضع عنوان الذاكرة الذي يشير إليه في مؤشر آخر على الجهة اليسرى فمثلاً.

```
Int x ;
// x متغير
int *p1 , *p2;
// p2 , P1 مؤشرين لرقم صحيح
p1 = & x ;
// x مؤشر لـ p1
/* x المؤشر p1 يشير إلى عنوان */
p2 = p1 ;
/* p1 المؤشر p2 يأخذ القيمة */
printf("%d, %d" , p1 , p2);
```

وبذلك يطبع البرنامج السابق قيم p2 , p1 التي هي في النهاية نفس القيم وكل منهما يشير إلى العنوان x في الذاكرة.

2 - العمليات الرياضية

لا تستطيع عمل أي عمليات حسابية على المؤشرات سوى الجمع والطرح ... ولكي يمكنك تخيل ما يحدث عند جمع وطرح المؤشرات أفترض أن p1 مؤشر لمتغير ذو قيمة 2000 وكما تعلم فإن المتغيرات من النوع الصحيح integers تكون ذات حجم 2 بايت. فإن استخدمت التعبير

```
p1++;
```

تصبح قيمة p1 = 2002 وليس 2001 لأنه سيشير إلى عنوان الذاكرة للرقم الصحيح التالي ... وهكذا في كل مرة يزيد فيها p1 بالأسلوب السابق يشير إلى المتغير الصحيح التالي 2004 ، 2006 ....

كذلك في حالة التناقص فإن المتغير

```
p1 -- ;
```

يؤدي إلى أن تكون محتويات p1 = 1998 وليس 1999 وذلك على اعتبار أنها كانت في البداية 2000. بمعنى أنه في كل مرة يتم تغيير المؤشر سواء بالزيادة أو النقصان فإنه يشير إلى موضع المتغير التالي من النوع الأصلي للمؤشر بزيادة أو نقصان تساوي حجم نوع البيانات التي يشير إليها المؤشر.

مثال

عند تطبيق مؤشر التزايد على مؤشر يستخدم للإشارة إلى حرف char فإنه يتزايد بمقدار 1 لأن char حجمه 1 بايت.

بينما عند تطبيق التزايد على مؤشر من النوع integer يزيد بمقدار 2 لأن integer يحتل 2 بايت من الذاكرة.

وليست العمليات الممكن تنفيذها على المؤشرات هي مجرد عمليات التزايد والتناقص وفقط لكن يمكن أن تضيف إلى أو تطرح من مؤشر قيم صحيحة مثل

```
p1=p1+5;
```

وذلك يعني أنك تجعل p1 يشير إلى العنوان الذي يلي العنوان الذي يشير إليه حالياً بـ 5 مواضع بحيث كل منها ذو حجم يماثل نوع البيانات الذي يشير إليه p1 .

وبخلاف ما سبق ليس هناك معنى لأي عمليات حسابية تتم على المؤشرات سواء بالضرب أو القسمة...

- **أي عمليات حسابية تتم على المؤشرات تعتمد على نوع البيانات التي يشير إليها المؤشر.**
- **لا يجوز إجراء عمليات الضرب والقسمة على المؤشرات**

3 - مقارنة المؤشرات

يمكنك مقارنة مؤشرين في أي تعبير كما يحدث في المتغيرات العادية فمثلاً التعبير التالي صحيح تماماً على الرغم من كون m , n مؤشرات لمواضع في الذاكرة.

```
If (m>n)printf("n pionts to lower memory than m\n");
```

المؤشرات والمصفوفات

هناك علاقة وثيقة بين المؤشرات والمصفوفات ذلك لأن الإشارة إلى عناصر المصفوفات تعد إحدى أهم وظائف المؤشرات فمثلاً

```
char star [50]; *p1;
// للإعلان عن مصفوفة star ومؤشر p1 كل منهما حرفي
p1 = star ;
```

```
pl يشير إلى العنصر الأول من المصفوفة على اعتبار أن//
ذكر star فقط بدون أقواس هو مؤشر للعنصر الأول //
```

من خلال السطرين السابقين تم الإعلان عن مصفوفة حروف حجمها 50 عنصر وكذلك الإعلان عن مؤشر pl يشير إلى char .

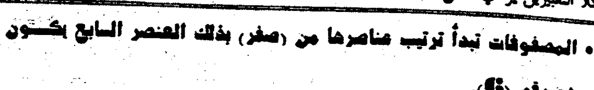
في السطر الثاني أصبح pl يشير إلى عنوان العنصر الأول في المصفوفة كذلك لو أردنا الوصول إلى قيمة العنصر السابع في المصفوفة يمكن كتابة

```
star[6];
```

أو تستخدم التعبير

```
*(pl+5);
```

كلا التعبيرين يؤدي نفس المهمة.



- **المصفوفات تبدأ ترتيب عناصرها من (صفر) بذلك العنصر السابع يكون هو رقم (6).**
- **استخدام اسم المصفوفة بدون تحديد عناصر وبدون أقواس يعطي مؤشر للعنصر الأول منها.**

وبذلك تتوافر في لغة C طريقتين للوصول إلى عناصر المصفوفة إحداها عن طريق تحديد العنصر بالطريقة العادية والثاني عن طريق المؤشرات ... وهو ما يعد أسرع كثيراً من الحالة الأولى.

ولأن السرعة تكون عامل مؤثر في تشغيل البرامج فإن الطريقة الثانية قد تكون أكثر شيوعاً في كتابة البرامج.

الإشارة إلى مصفوفات الأحرف

كما نعلم تتعامل لغة C مع النصوص من خلال مصفوفة من الأحرف، لأن مثل هذه المصفوفة يتم التعامل معها بشكل تتابعي فإن المؤشرات تكون أنسب كثيراً في مثل هذه الحالة للإشارة إلى أي من عناصر المصفوفة على سبيل المثال.

مصفوفة المؤشرات

كأي نوع بيانات أخر يمكن تكوين مصفوفة جميع عناصرها من المؤشرات فمثلاً تستطيع الإعلان عن مصفوفة لها 10 عناصر كل منها يشير إلى رقم صحيح integer بالجملة.

```
Int *star[10];
```

وبعدها نتعامل مع المصفوفة بالشكل العادي كما يلي

```
star[6] = &z;
```

لجعل العنصر السادس من المصفوفة يشير إلى عنوان المتغير z وبالعكس فإن z تستطيع إعطاءها قيمة كالتالي

```
z = *star[3];
```

لوضع القيمة التي يشير إليها العنصر الرابع من المصفوفة في المتغير z وهو ما كان يحـــدث بـــالضبط مــع المتغيرات والمصفوفات العادية.

كذلك أن احتجت إلى تمرير مصفوفة من المؤشرات لدالة ما فإنك تستخدم نفس الأسلوب القديم المستخدم مع المصفوفات العادية ، فقط تستدعي الدالة باستخدام اسم المصفوفة بدون أي رقم للعناصر داخلها علـــى سبيل المثال الدالة التي تستخدم المصفوفة star السابق الإعلان عنها ربما تكون كالتالي:

```
void display-array (int*star[]) /* الشكل المبدئي
للدالة */
{
int t;
for(t=0; t<10, t++)
printf("%d" , *star[t]);
}
```

ونلاحظ من خلال السطر الأول أننا أعلنا الدالة بحيث تأخذ وسيطات int *star[] وبالتالي فهي ليســت مجرد مؤشر لـــ interger لكنها مؤشر إلى مصفوفة عناصرها تشير إلى interger وبذلك يمكن الإشارة إلى جميع عناصر المصفوفة ثم طباعتها باستخدام دوارة for كما سبق مع المصفوفة العادية.

ومن أهم استخدامات مصفوفات المؤشرات مجال التعامل مع رسائل الخطأ كما في المثال التالي:

```
void error(int num)
// دالة تتحكم في طباعة رسائل الخطأ رقم الخطأ
{
static char *err[] = {     الاعلان عن المصفوفة وبدئها في نفس الجملة
"can not opn file\*n" ;    العنصر الأول
" write error \n";         العنصر الثاني
"read error\n" ;           العنصر الثالث
};                         نهاية المصفوفة
printf("%s", err[num]);
}
```

وكما ترى بأن دالة printf() استدعيت من خلال مؤشر لحـرف char pointer مـن داخـل الدالـة error().

والمؤشر يشير إلى واحدة من الرسائل الموجودة في المصفوفة على حساب الرقم num الذي تأخذه الدالة في البداية ويتحدد على أساسه أي عناصر المصفوفة يتم طباعته.

ومن الجدير بالذكر أن هناك بعض الأحوال التي قد تحتاج فيها للإشارة إلى مؤشر بمعنى إيجاد متغـير يحمـل عنوان مؤشر في الذاكرة ويتم ذلك باستخدام المؤشر(*) كالتالي:

```
int i ;
int *p
int **q;
    // يجب الإعلان عن q بهذا الشكل //
i = 15;
p = &i ;
        // i لـــ مؤشر p
q=&p
       // p لـــ يصبح مؤشر q
```

مزايا استخدام المؤشرات pointer

يحقق استعمال المؤشرات فوائد كثيرة منها

- اعادة أكثر من قيمة من الدوال.
- التعامل مع المصفوفات والحرفيات وتمريرها الى الدوال بشكل أفضل
- انشاء أنواع أكثر قوة من البيانات
- التعامل مع الجهاز ومكوناته وعناوين مداخل ومخارج الجهاز

اعادة أكثر من قيمة من الدوال

من الفوائد المشهورة للمؤشرات استخدامها فى اعادة أكثر من قيمة من الدالـة. فما معنى ذلك ؟

فى الفصل الخامس شرحنا الدوال والتعامل معها وكيفية اعـادة قيمة من الدالة لاحظنا فى الأمثلة التى استخدمناها أننا استخدمنا كلمة return مرة واحدة مـع كـل دالـة وهذا معناه عدم امكانية اعادة أكثر من قيمة من الدالة.

فلو فرضنا أن لدينا مجموعة عمليات وأردنا انشاء دالة لهـذه العمليـات وانشـأنا الدالة وتم حساب نتائج العمليات ووضعت هذه النتائج فـى متغيرات وأردنا أعـادة هـذه القيم الى الدالة الرئيسية ، هنا تظهر المشكلة. وهى أننـا لا نسـتطيع اسـتعمال أكـثر مـن كلمة return وكلمة return لا تعيد الا قيمة واحدة أما فى حالة استخدام المؤشـرات فيمكننا إعادة أكثر من قيمة.

ولتوضيح ذلك سـنكتب برنامجـان البرنـامج الأول بـدون اسـتعمال المؤشـرات وفيه ستظهر هذه المشكلة ، والبرنامج الثانى باستخدام المؤشـرات ومنـه سـتعرف كيـف يمكن حل هذه المشكلة

يشتمل الشكل رقم ١-٨ على برنامج يقوم بانشاء دالة تأخذ معاملين من نوع صحيح ثم تقوم الدالة باضافة القيمة ٥ الى كل معامل

```
0:  /* Program Name CS8_1.C*/
1:  #include <stdio.h>
2:  #incldue <conio.h>
3:  void get2(int xx, int yy);
4:  main ()
5:    {
6:       int x=4,y=7;
7:       get2(x,y);
8:       printf("first no:is %d secand no:%d",x,y);
9:     }
10: void get2(int xx,int yy)
11:  {
12:      xx+=5;
13:      yy+=5;
14:  }
```

الشكل رقم ١-٨ انشاء دالة داخل البرنامج

وعن هذا البرنامج نوضح مايلي :

يبدأ البرنامج الموجود في الشكل رقم ١-٨ في السطر رقم ٣ بالاعلان عن الدالة () get2 التي تأخذ معاملان من نوع صحيح وفي السطر رقم ٧ يتم استدعاء الدالة مع ارسال قيمتين صحيحتين لها تأخذهما الدالة الى السطر رقم ١٠ وتضعهما في المتغيرين xx,yy.

وفي السطرين رقم ١٢و١٣ يتم اضافة القيمة ٥ الى كل من المتغيرين ثم تقوم الدالة الرئيسية في السطر رقم ٨ بطباعة قيم المتغيرين.

والسؤال هنا ما هي القيم التي يطبعها البرنامج؟ هل يطبع البرنامج القيم بعد اضافة القيمة ٥ الى كل متغير كما في سطور الدالة أما يطبعها كما هي ؟.قبل أن تتابع شرح البرنامج حاول الوقوف والتفكير في ذلك.

قد تظن للوهلة الأولى أن النتيجة هي ٩ و ١٢ وذلك بعد اضافة القيمة ٥ الى القيمتين ٤ ، ٧ في حين أنك لو دققت النظر ستجد أن الدالة من النوع void كما في الاعلان في السطر رقم ٣ وبالتالي الدالة لاتعيد قيم وهذا ماتم حيث قامت الدالة باستقبال القيم واضافة القيمة ٥ الى كل عنصر ، ولكن لم تعيد الدالة النتيجة وبالتالي تظل قيم المتغيرات كما هي ٤و٧ وحتى لو أعلنا أن نوع الدالة int فلن تستطيع الدالة اعادة أكثر من قيمة. اذا كيف نعيد قيم المتغيرين في البرنامج السابق بعد اضافة القيمة ٥ اليهما هذا مانراه من خلال البرنامج الموجود في الشكل رقم ٢-٨.

```
0:  /*Program Name CS8_2.C*/
1:  #include <stdio.h>
2:  #include <conio.h>
3:  void rets(int *xx,int *yy);
4:  main ()
5:    {
6:      int *x=5,*y=10;
7:      rets(&x,&y);
8:      printf ("first no:is %d secand is %d",x,y);
9:    }
10: void rets (int *xx,int *yy)
11:   {
12:     *xx+=5;
13:     *yy+=10;
14:   }
```

شكل رقم ٢-٨ استخدام المؤشرات مع الدوال

وعند تنفيذ البرنامج تحصل على النتيجة التالية

first no:is10 secand is 20

وعن هذا البرنامج نوضح ما يلي :

في السطر رقم ٣ اعلان عن دالة لها معاملان وهذان المعاملان من نوع مؤشر الى قيمة صحيحة ، ومعنى أن المعاملات من نوع مؤشرات أننا في حالة استدعاء الدالة لن نرسل الى الدالة قيم ولكن نرسل الى الدالة عناوين هذه القيم ، وهذا ما تم في السطر رقم ٧ حيث تم ارسال عناوين المتغيران x ,y وذلك بالصورة &x,&y فوجود العلامة & مع المتغير يجعل المتغير يشير الى عنوان المكان وليس القيمة المخزنة في المتغير ففي هذا السطر يتم ارسال عناوين المتغيرين x,y الى الدالة rets() التي تقوم باستقبال العناوين والتعويض بهما في المتغيرين xx,yy ،وتقوم الدالة بزيادة القيم الموجودة في هذه العناوين. أي أن ارسال العناوين يجعل الدالة تتعامل مع القيم الموجودة في هذه العناوين وبالتالي تكون نتيجة هذا البرنامج هي طباعة القيم بعد زيادة القيمة المخزنة في المتغير. وبهذا الاسلوب كأننا اعادنا قيمتين من الدالة. هل لاحظت فكرة ارسال عنوان الى دالة اذا ما فائدة هذه الفكرة ؟

الفائدة هي امكانية انشاء دالة تقوم بعمليات كثيرة وتخرج أكثر من ناتج أما بدون استعمال المؤشرات فلا تستطيع ان تعيد هذه القيم الى الدالة الرئيسية لانك لا تستطيع استعمال أكثر من جملة return.

ولكن يمكن كما في هذا المثال أن نستدعي الدالة بحيث نرسل لها عناوين أي عدد من المتغيرات والدالة بدورها تجرى العمليات المطلوبة ثم تضع الناتج في هذه العناوين ونستخدمها نحن من داخل الدالة الرئيسية.

✍ للاعلان عن مؤشر تضع العلامه * قبل المتغير وللتعامل مع عنوان المكان مع الدالة يسبق المتغير بالعلامه & فمثلاً نكتب p* للاعلان عن مؤشر حسب النوع

ونكتب p& لإرسال عنوان المكان

المؤشرات والمصفوفات Pointers and Arrays

للمؤشرات دور مع المصفوفات حيث تتعامل مع عناوين عناصر المصفوفات ، وهذا بالطبع أسرع من الطريقة المعتادة. وقبل أن نوضح كيف تتعامل المؤشرات مع المصفوفات نورد مثالاً يذكرنا بالتعامل مع المصفوفات بدون استعمال المؤشرات. والبرنامج الموجود بالشكل رقم ٣-٨ يقوم بالاعلان عن مصفوفة واعطائها قيم ابتدائية ثم يقوم بطباعة هذه القيم على الشاشة.

```
0:  /*Program Name CS8_3.C*/
1:  #include <stdio.h>
2:  #include <conio.h>
3:    main ()
4:     {
5:         int nums[ ]={92,81,70,69,58};
6:         int dex;
7:         for (dex=0;dex<5;dex++)
8:             printf ("\t%t",nums[dex]);
9:     }
```

شكل رقم ٣-٨ التعامل مع المصفوفة بدون مؤشرات

وعند تنفيذ هذا البرنامج نحصل على النتيجة التالية

```
92      81      70      69      58
```

اما فى حالة استخدام المؤشرات مع المصفوفات فيتم التعامل مع عناصر المصفوفة عن طريق اسم المصفوفة ، حيث يعتبر اسم المصفوفة هو عنوان أول عنصر فيها فمثلا المصفوفة int a [10] يعتبر اسم المصفوفة وهو a هو عنوان أول عنصر. فاذا طبعنا قيمة المتغير a نحصل على عنوان أول عنصر فى المصفوفة ولطباعة قيمة أول عنصر نسبق اسم المصفوفة بالعلامة * وبالتالى الصورة a* تعبر عن قيمة أول عنصر واذا

أضفنا ١ الى عنوان المصفوفة ستحصل على عنوان ثانى عنصر فمثلاً (a+1)* تشير الى قيمة ثانى عنصر وهكذا. والبرنامج الموجود بالشكل رقم (٤-٨) يوضح ذلك

```
0:  /*Program Name CS8_4.C*/
1:  #include <stdio.h>
2:  #include <conio.h>
3:   main ()
4:    {
5:         int nums[ ]={92,81,70,69,58};
6:         int dex;
7:         for (dex=0;dex<5;dex++)
8:             printf ("\t%d",*(nums+dex));
9:    }
```

الشكل رقم ٤-٨ استخدام المؤشرات مع المصفوفات

وعن هذا البرنامج نوضح ما يلى :

- فى السطر رقم ٥ اسم المصفوفة هو nums وهو فى نفس الوقت عنوان أول عنصر كما اشرنا بمعنى لو كتبت السطر ;printf("%d",nums) سوف يطبع لك عنوان أول عنصر تم حجزه لهذه المصفوفة حيث أنه فى حالة الاعلان عن مصفوفة يتم حجز أماكن بعدد العناصر ولكن يشار الى أول عنصر فقط باسم المصفوفة وتميز نهاية المصفوفة العلامة '\0' فإذا وضعنا العلامة * بجانب numsبالشكل nums* ؟ فإننا نشير الى القيمة الموجودة بهذا العنوان وبالتالى الصورة ;printf ("%d",*nums) تطبع قيمة أول عنصر.
- والصورة ;("%d",nums+1) تطبع عنوان العنصر الثانى
- والصورة ;("%d",*(nums+1)) تطبع قيمة العنصر الثاني
- والصورة ;printf ("%d",nums+2) تطبع عنوان العنصر الثالث
- والصورة ;printf ("%d",*(nums+2)) تطبع قيمة العنصر الثالث وهكذا

معنى السجل (STRUCTURE) والحاجة الى استعماله

من أهم التطبيقات في عالم البرامج تطبيقات قواعد البيانات فمثلاً قاعدة بيانات موظفين تمثل بيانات الموظفين في شكل سجلات كل سجل يتكون من مجموعة حقول ولو أن لك خبرة بأحد برامج قواعد البيانات مثل dbase فستعرف أن الملف ينقسم الى سجلات (records) والسجل ينقسم الى حقول (fields) ودائما نحتاج للتعامل مع السجل كوحدة وكذلك مع الحقول كوحدة. وتستخدم لغة C كلمة Stucture بنفس المفهوم الذى تستخدمه لغات البرمجة الأخرى لكلمة Record .

استعمال السجل (STRUCTURE)

هناك خطوات تتبع للتعامل مع السجل وهى انشاء السجل (تركيب السجل) وتحديد الحقول المطلوبه ثم الاعلان عن متغير من نوع هذا السجل ثم التعامل مع حقول هذا السجل. والبرنامج الموجود بالشكل ١-٩ يشتمل على هذه الخطوات

```
0:  /*Program Name CS9_1.C
1:  #include  <stdio.h>
2:  #include  <conio.h>
3:  main ()
4:    {
5:       struct data
6:                  {
7:                  int num;
8:                  char stat ;
9:                  };
10:
11:       struct data stud;
12:       stud.num=5;
13:      stud.stat 't';
```

```
14:     printf ("\n stud.num=%d,stud.stat=%c",stud.num,stud.stat);
15: }
```

الشكل ١-٩ برنامج إنشاء السجل و إستخدامه

عن هذا البرنامج نوضح مايلى :

- فى السطر رقم ٥ يبدأ انشاء السجل وذلك باستعمال كلمة struct واعطاء هذا السجل اسم وهو data وكلمة data ممكن أن تكون أى كلمة.
- فى السطر رقم ٦ تبدأ مكونات هذا السجل بالقوس } وفى السطرين رقم ٧ ورقم ٨ اعلان عن حقول السجل وهى عبار عن متغير من نوع صحيح ومتغير من نوع حرف وينتهى السجل فى السطر رقم ٩ بالقوس {
- فى السطر رقم ١١ يتم الاعلان عن متغير من نوع السجل وهو المتغير stud وبالتالى أخذ المتغير stud نفس التركيب فأصبح له عنصر اسمه num من نوع صحيح وكذلك عنصر من نوع حرف وهو stat.
- فى السطرين رقم ١٢ ورقم ١٣ تم اعطاء قيم لحقول السجل ولكن الملاحظ أنه للتعامل مع حقل فى سجل يتم كتابة اسم الحقل مسبوقا باسم السجل التابع له وبينهما نقطة بالصورة stud.num
- فى السطر رقم ١٤ يتم طباعة قيم حقول السجل وبنفس الاسلوب تم كتابة اسم الحقل مسبوقا باسم السجل وبينهما النقطة للاشارة أن هذا الحقل تابع لهذا السجل.

وعند تنفيذ هذا البرنامج نحصل على النتيجة التالية :

stud.num=5,stud.stat=t



يمكن الاعلان عن أكثر من متغير من نوع السجل كما يحدث مع أنواع البيانات الاخرى فمثلا أنظر الاعلان التالى struct data stud1,stud2,stud3 ومعناه أن المتغيرات stud1,stud2,stud3 من نوع data وبالتالى لها نفس العناصر.

وكذلك يمكن الاعلان عن متغيرات من نوع structure بطريقة أخرى نوضحها فيما يلى :

```
struct data
        {
        int num;
        char ch;
        }stud1,stud2;
```

ويعطى هذا الاعلان نفس النتيجة السابقة حيث أصبح stud1,stud2 لهما نفس تركيب السجل.

كيفية ادخال بيانات الى عناصر السجل structure

يمكن معاملة عناصر السجل structure معاملة المتغيرات العادية حيث يمكن اعطائها قيم كما سبق ويمكن استقبال قيم بدوال الاستقبال من المستخدم ووضع هذة القيم فى عناصر السجل والبرنامج الموجود فى الشكل رقم ٢-٩ يقوم بانشاء سجل واستقبال قيم عناصره من المستخدم ثم طباعة هذه القيم على الشاشه

```
0:  /* Program Name CS9_2.C*/
1:  #include <stdio.h>
2:  main ()
3:      {
4:        struct data
5:            {
6:              int no;
7:              char name[10];
8:            };
9:      struct data stud;
10      printf("\n\n  stud.no=");
11:     scanf("%d",&stud.no);
12:     printf("\n\n  stud.name=");
13:     scanf("%s",stud.name);     /*Notes name is string*/
```

```
14:   clrscr();
15:   printf("\n\n stud.no=%d",stud.no);
16:   printf("\n\n  stud.name=%s",stud.name);
17: }
```

شكل رقم ٢-٩ إنشاء السجل و إستقبال عناصره ثم طباعتها

وعن هذا البرنامج نوضح مايلى

- من السطر رقم ٤ الى السطر رقم ٨ تم انشاء السجل
- فى السطر رقم ٩ تم الاعلان عن متغير من نوع السجل
- فى السطر رقم ١١ استخدمنا دالة الاستقبال ()scanf لاستقبال عناصر السجل وهنا يتضح الفرق بين التعامل مع متغير عادى ومتغير عنصر فى سجل وهو أننا ننسب عنصر السجل الى السجل التابع له وذلك عن طريق النقطة. بالصورة stud.no ومعناها الاشارة الى العنصر no التابع للسجل المسمى stud ويتم التعامل مع عناصر السجل كما يتم التعامل مع المتغيرات العادية.

وعند تنفيذ هذا البرنامج نحصل على النتيجة التالية:

```
stud.no=1
stud.name=mohamed
stud.no=1
stud.name=mohamed
```

وضع محتويات سجل فى آخر

درسنا من قبل امكانية مساواة متغيرين من نوع واحد وذلك لوضع قيمة المتغير الأول فى المتغير الثانى وذلك بالصورة التالية :

```
int a,b;
b=5;
a=b;
```

وهذا معناه وضع القيمة المخزنة فى المتغير b وهى القيمة 5 فى المتغير a.
ويمكن تحقيق ذلك مع السجلات بحيث يمكن مساواة متغير من نوع سجل مع أخر وبالتالى يتم مساواة قيم جميع العناصر بين السجلين بشرط أن يكون السجلين من نفس النوع، والبرنامج الموجود بالشكل ٣-٩ يوضح ذلك

```
0:  /* Program Name CS9_3.C*/
1:  #include  <stdio.h>
2:  #include  <conio.h>
3:  main ()
4:    {
5:        struct data
6:          {
7:               int num;
8:               char na;
9:          };
10:
11:    struct data stud1,stud2;
12:    stud1.num=5;
13:    stud1.na='t';
14:    stud2=stud1;
15:printf\("\n stud1.num=%d,stud1.na=%s",stud1.num,stud.na);
16:printf("\n stud2.num=%d,stud2.na=%s",stud2.num,stud.na);
17   }
```

شكل ٣-٩ وضع محتويات سجل فى سجل آخر

وعن هذا البرنامج نوضح مايلى :

- فى السطر رقم ٥ تم انشاء سجل جديد
- فى السطر رقم ١١ تم تعريف متغيرين من نوع السجل هما stud1,stud2
- فى السطر رقم ١٤ تم مساواة المتغيرين stud1 و stud2 وبالتالى وضع نسخة من القيم الموجودة فى السجلstud1 فى السجل الثانى stud2.

وعن تنفيذ هذا البرنامج نحصل على النتيجة التالية :

```
stud1.num=5,stud1.na=t
stud2.num=5,stud2.na=t
```

السجلات المتداخلة Nested Structures

شرحنا أن السجل هو مجموعة من العناصر أيا كان نوع هذه العناصر وبالتالى يمكن أن تكون العناصر أو بعضها سجلات وهذا ما يسمى بالسجلات المتداخلة والبرنامج الموجود بالشكل رقم ٤-٩ يوضح كيف يكون السجل عنصر فى سجل آخر وكيفية التعامل مع عناصر السجلات فى هذه الحالة.

```
0:  /* Program Name CS9_4.C*/
1:  #include <stdio.h>
2:  main ()
3:    {
4:      struct person
5:        {
6:          int no;
7:          char name[10];
8:        };
9:      struct  group
10:       {
11:         struct person peno1;
12:         struct person peno2;
13:         int code;
14:       };
15:     struct group group1;
16:     printf("\n\n  group1.peno1.no=");
17:     scanf("%d",&group1.peno1.no);
18:     printf("\n  group1.peno1.name=");
19:     scanf("%s",group1.peno1.name);
20:     printf("\n group1.code=");
21:     scanf("%d",& rcup1.code);
```

```
22:    group1.peno2=group1.peno1;
23:    clrscr();
24:    printf("\n\n the data of groups:\n\t");
25:    printf("\n group1.peno1.no=%d",group1.peno1.no);
26:    printf("\n group1.peno1.name=%s",group1.peno1.name);
27:    printf("\n group1.code=%d",group1.code);
28:    printf("\n group1.peno2.no=%d",group1.peno2.no);
29:    printf("\n group1.peno2.name=%s",group1.peno1.name);
30:  }
```

شكل رقم ٤-٩ السجلات المتداخلة

وعن هذا البرنامج نوضح ما يلى :

فى هذا البرنامج تم الاعلان عن سجل يمثل بيانات اشخاص ثم تم الاعلان عن سجل أخر والعنصر الثانى فى هذا السجل هو سجل من نوع السجل الأول وأخذ الأسم peno1 ثم تم التعامل مع عناصر السجلات كما فى سطور البرنامج كما يلى

- من السطر رقم ٤ الى السطر رقم ٨ تم انشاء السجل الأول وهو person وعناصره هى no و name ومن السطر رقم ٩ الى السطر رقم ١٤ تم انشاء السجل الثانى هو group وعناصره هى code ويمثل كود المجموعة والسطر رقم ١٥ اعلان عن متغير من نوع السجل الثانى
- من السطر رقم ١٦ الى السطر رقم ٢٢ يتم استقبال العناصر ومن السطر رقم ٢٤ الى السطر رقم ٢٩ يتم طباعة عناصر السجل مع ملاحظة أننا ننسب كل عنصر الى السجل التابع له

وعند تنفيذ هذا البرنامج نحصل على النتيجة التالية :

```
group1.peno1.no=1
group1.peno1.name=hamdy
group1.code=01
the data of group:
```

```
group1.peno1.no=1
group1.peno1.name=hamdy
group1.code=01
group1.peno2.no=1
group1.peno2.name=hamdy
group1.code=01
```

السجلات والدوال

وهنا نناقش

- استعمال السجل كمعامل للدالـــة
- اعادة السجل من الدالـــة

درسنا فى فصل الدوال أن الدالة يمكن أن تستعمل معاملات من أى نوع من البيانات وبالتالى يمكن أن يكون هذا المعامل من نوع سجل

بمعنى انه يمكن أن نرسل للدالة سجل كمعامل والدالة بدورها تقوم بأى عمليات على هذا السجل وبالمثل يمكن للدالة أن تقوم ببعض العمليات على السجل ثم تعيد سجل الى الدالة الرئيسية ويكون نوع الدالة من نوع هذا السجل والبرنامج الموجود بالشكل ٥-٩ يقوم بالاعلان عن دالتين الاولى تقوم باستقبال عناصر سجل من المستخدم وعند استدعائها تعيد هذا السجل الى الدالة الرئيسية والدالة الثانية تأخذ هذا السجل كمعامل وتقوم بطباعة بياناته على الشاشة

```
0:   /*Prgram Name CS9_5.C*/
1:   struct personal
2:       {
3:          char name[30];
4:          int numb;
5:       };
6:    struct personal addname(void);
6:    void display(struct personal  custm);
7:    main ()
```

```
8:    {
9:         struct personal custmer1;
10:        struct personal custmer2;
12:        custmer1=addname();
13:        custmer2=addname();
14:        display(custmer1);
15:        display(custmer2);
16:   }
17:      /* addname()  */
18:    struct personal addname()
19:        {
20:             char numstr[81];
21:             struct personal custmer;
22:             printf ("\n new custmer \n Enter name:");
23:             gets(custmer.name);
24:             printf ("\n Enter custmer no:");
25:             gets(numstr);
26:             custmer.numb=atoi(numstr);  /*convert string to
                                                          integer*/
27:             return custmer;
28:         }
29:     /* list() */
30:   void display(struct personal custm)
32:     {
33:        printf("\n Custmers:\n");
34:        printf ("Name:%s\n",custm.name);
35:        printf ("Number:%d \n",custm.numb);
      }
```

شكل ٥-٩ السجلات والدوال

وعن هذا البرنامج نوضح مايلى :

فى السطر رقم ٦ تم الاعلان عن الدوال ()addname و ()display .والدالـة ()addname من نوع struct personal فماذا يعنى أنها من هذا النوع ، نعرف أن نوع الدالة يعتمد على نوع القيمة التى تعيدها الدالة ولو نظرت الى كلمة return فى الدالـة ()addname فى السطر رقم ٢٧ تجد أنها تعيد structure من نوع struct

personal لذلك لابد أن يكون نوعها مـن نفـس نـوع القيمـة التـى تعيدهـا وهـو struct personal. والدالة display لا تعيد قيم بل تقوم بطباعة قيم عناصر السـجل structure فقط لذلك كان نوعها void

وعند تنفيذ هذا البرنامج نحصل على النتيجة التالية :

```
new custmer
Enter name:samy
Enter custmer no:1
new custmer
Enter name:hamdy
Enter custmer no:2
Custmer:
Name:samy
Number:1
Custmer:
Name:hamdy
Number:2
```

مصفوفة السجلات Arrags Of Structures

درسنا فى درس المصفوفات أن المصفوفة هى مجموعة مـن العناصر مـن نفـس النوع هذا النوع يمكـن أن يكـون أى نـوع مـن البيانـات وبالتـالى يمكـن أن نعلـن عـن

مصفوفة عناصرها من نوع سجلات والبرنامج الموجود بالشكل ٦-٩يقوم بانشاء سجل ثم الاعلان مصفوفة من هذا السجل ثم استقبال قيم عناصر هذه المصفوفة ثم طباعتها

```
/* Prgram Name CS9_6.C*/
#include <stdio.h>
main ()
  {
   int i;
  struct personal
      {
      char name[30];
      int numb;
      };
  struct personal custmer[5];
  for(i=0;i<5;i++)
     {
       printf("\n custmer.no[%d]=",i);
       scanf("%d",& custmer[i].numb);
       printf("\n custmer.nname[%d]=",i);
       scanf("%d", custmer[i].name);
     }
  for(i=0;i<5;i++)
     {
       printf("\n %d \t %s", custmer[i].numb, custmer[i].name);
     }
```

شكل ٦-٩ إستخدام مصفوفة السجلات

المؤشرات والسجلات Pointers and Structures

ذكرنا أن المؤشرات (pointers) تدخل فى كل عناصر اللغة وبالتالى لها دور مع السجلات (structures) وهنا نناقش كيفية الاعلان عن مؤشر من نوع structure (pointer to structure) وكيفية استعمال هذا المؤشر مع السجلات (structures).

الفرق بين اعلان مؤشر الى سجل واعلان متغيرمن نوع سجل هو أن المؤشر يسبق بالعلامة * ونوضح ذلك من خلال البرنامج الموجود بالشكل رقم ٧-٩

```
0:  /* Program Name CS9_7.C*/
1:  void main (void)
2:    {
3:      struct xx
4:        {
5:          int num1;
6:          char ch;
7:        };
8:     struct xx xx1;
9:     struct xx *ptr;
10:
11:     ptr =&xx1;
12:     ptr->num1=303;
13:     ptr->ch='q';
14:     printf ("ptr->num1=%d\n",ptr->num1);
15:     printf ("ptr->ch=%c",ptr->ch);
16:    }
```

شكل رقم ٧-٩ إستخدام المؤشرات مع السجلات

وعن هذا البرنامج نوضح ما يلى

فى السطر رقم ٣ بدأ البرنامج بانشاء سجل ، وفى السطر رقم ٨ تم الاعلان عن متغير من نوع هذا السجل ، وفى السطر رقم ٩ تم الاعلان عن مؤشر الى هذا السجل وفى السطر رقم ١١ تم تخزين عنوان السجل xx1 فى المتغير ptr وبعد ذلك يتم التعامل مع السجل عن طريق التعامل مع المؤشر الى هذا السجل كما فى السطر رقم ١٢ والسطر رقم ١٣ ونلاحظ أن التعامل مع عنوان السجل وعناصر السجل هو نفس التعامل مع متغير السجل وعناصره غير أننا نستبدل الصورة xx.no بالصورة ptr->no حيث أن ptr عبارة عن مؤشر إلى السجل

وعند تنفيذ هذا البرنامج نحصل على النتيجة التالية

```
ptr->num1=303
ptr->ch=q
```

تغيير نوع البيانات TYPECASTING

من العمليات المفيدة فى لغة C وجود أكثر من طريقة لتحويل بيانات من نوع الى نوع آخر وتتم بطريقتين:

الطريقة الاولى : استخدام مجموعة دوال مثل الدالة atoi() بمعنى ascii to int وهى تقوم بتحويل الارقام التى فى صورة حرفيات الى صورة أرقام للتعامل معها كأرقام والدالة atof() بمعنى ascii To float والتى ذكرناها فى فصل المصفوفات.

الطريقة الثانية : هى ما يسمى Typecasting وهى كتابة النوع المطلوب التحويل اليه قبل المتغير المطلوب تغير نوعه. والبرنامج الموجود فى الشكل رقم ٨-٩ يوضح فكرة تغير النوع باستخدام هذه الطريقة

```
0:  /* Prgram Name CS9_8.C*/
1:  #include<stdio.h>
2:  main ( )
3:    {
4:      int a=66,b;
5:      char ch='t',g;
6:       b=(int) ch;
7:        g=(char) a;
8:        printf("b=%d",b);
9:        printf("\n g=%c",g);
10:   }
```

شكل رقم ٨-٩ برنامج تغيير نوع البيانات

الماكرو (MACROS)

ما المقصود بالماكرو ؟

هو مجموعة تعليمات تؤدى غرض معين ويشبه الى حد كبير الدالة ،ويتم انشائه مرة واحدة وبعد ذلك يمكنك استدعائه كلما احتجت اليه.

وقبل أن نسأل ما الفرق بينه وبين الدالة تعال بنا نرى أولا كيفية انشائه واستعماله ثم نناقش بعد ذلك الفرق ثم نوضح متى نستخدم الماكرو ومتى نستخدم الدالة.

كيفية انشاء الماكرو

يتم ذلك باستعمال الكلمة #define وهذه الكلمة تسمى directive أو preprocessor ومعناها توجيه.

ولانشاء الماكرو تستخدم الصورة التالية :

```
#define macro line
```

وهى عبارة عن تعريف طرف بطرف مثل #define A 5 ومعناها عرف المتغير A بالقيمة 5

والبرنامج الموجود بالشكل رقم (١١-٥) يوضح لنا كيفية الاعلان عن الماكرو وكيفية إستعماله

```
0:  /*CS5_11.C*/
1:  #define sum(a,b) a+b
2:  #define mul(x,y) x*y
3:  #include <stdio.h>
4:  main ()
5:     {
6:       int v1=5,v2=10;
7:       printf("\n\n sum(v1,v2)=%d",sum(v1,v2));
8:       printf("\n\n mul(v1,v2)=%d",mul(v1,v2));
9:     }
```

الشكل ١١-٥ الإعلان عن الماكرو واستعماله

وعند تنفيذ هذا البرنامج ستحصل على النتيجة التالية :

```
sum(v1,v2)=15
mul(v1,v2)=50
```

وعن هذا البرنامج نوضح ما يلى :

في السطر رقم ١ استخدمنا كلمة define لتعريف ماكرو بالاسم sum ووظيفته استبدال المتغيرين a,b بالصورة a+b وبالمثل في السطر رقم ٢ يستبدل المتغيرين x,y بنتيجة الضرب x*y ومعناها كلما قابل مترجم اللغة الطرف الأول للماكرو يستبدله بالطرف الثاني.

ونلاحظ في هذا المثال أن المتغيرين a,b يمكن استبدالهما بأى متغيرين أو قيمتين داخل البرنامج واسم الماكرو هو الذى يحدد العملية التى يقوم بها الماكرو هل هى عملية جمع أم ضرب بناءً على المعادلة الموجودة فى الطرف الايمن من الماكرو

في السطر رقم ٧ استخدمنا الماكرو sum لجمع متغيرين هما v1,v2 وبالمثل السطر رقم ٨.

ما الفرق بين الماكرو وبين الدالة ومتى نستخدم الماكرو ومتى نستخدم الدالة؟

من المعروف أن التعامل مع البرنامج يمر بثلاث مراحل :

١. كتابة البرنامج وهذا ما نسميه source code ويخصص لملف المصدر الامتداد C.
٢. ترجمة البرنامج للغة يفهمها الحاسب ويسمى compilation ويخصص للملف الامتداد obj.
٣. ربط ملف object بمكتبات اللغة ليصبح قابل للتنفيذ وتسمى هذه العملية linking ويخصص للملف امتداد exe.

ويظهر الفرق بين الماكرو وبين الدالة في هذه المراحل كما يلي

في مرحلة الكتابة ليس هناك فرق بين الماكرو والدالة .في مرحلة الترجمة (compilation). يتم تحويل تعليمات الدالة الى لغة الالة (object) وتنتظر مرحلة الربط ولاتنفذ الدالة الا في مرحلة الربط. أما في حالة الماكرو يتم استبدال الماكرو بنتيجة تنفيذ الماكرو فبالرجوع الى البرنامج الموجود بالشكل رقم ١١-٥ يتم استبدال الماكرو الموجود بالسطر رقم ٧ بنتيجة التنفيذ مباشرة أى يتم وضع القيمة ١٥ التي هي نتيجة تنفيذ الماكرو مكان sum(v1,v2) وبالتالي عندما تأتي مرحلة التنفيذ يجد البرنامج نتيجة تنفيذ الماكرو وهي ١٥ وبالمثل يمكن أن تفهم الماكرو الموجود في السطر رقم ٨

وهكذا نرى أن للماكرو مزايا منها :

أنه بسيط فى الانشاء بسيط فى الاستعمال ويعطى فى النهاية ملف تنفيذى أصغر من المقابل له باستعمال الدوال فإذا كانت العملية المطلوبة عمل داله لها بسيطة ويمكن كتابتها فى سطر واحد نستعمل الماكرو أما إذا كانت تحتاج أكثر من سطر نستخدم الدالة.

المشروع Project

إذا كانت لك خبرة بقاعد البيانات DBASE أو CLIPPER ، فأنت تعلم أن انشاء تطبيق متكامل يتم عن طريق عمل ملف رئيسى (برنامج رئيسى) وبرامج فرعية.

ويتم استدعاء البرامج الفرعية من البرنامج الرئيسى وذلك من داخل البرنامج الرئيسى بالأمر do أى تقسيم البرنامج الى برنامج رئيسى وبرامج فرعية .

والسؤال كيف يتم ذلك فى لغة C ؟

يتم ذلك بأكثر من طريقة

اذا كان البرنامج الكلى صغيرا ولا يتطلب أكثر من ملف يكفى استعمال الدوال كما مر بنا فى البرامج السابقة أى ننشئ الدالة الرئيسية ()main وبداخلها يتم استدعاء الدوال الاخرى.

اذا كان البرنامج كبير بحيث يتطلب أكثر من ملف فهناك طريقتين لتنظيم ذلك

الطريقة الاولى هى طريقة تقليدية اجتهادية لاتستعمل كثيرا وهى أن نكتب البرنامج الرئيسى وبه الدالة الرئيسية فى ملف منفرد ونكتب البرامج الفرعية فى صورة دوال ونضع هذه الدوال فى ملف منفصل. ويتم استدعاء هذه الدوال من داخل الدالة الرئيسية ولكى يتم ربط الملف الثانى الذى به الدوال بالملف الاول نكتب اسم الملف الثانى مع

كلمة #include بالصورة <file2.c> #include في بداية الملف الاول ولتوضيح ذلك نسوق المثال التالي :

في هذا المثال ستجد ملفين الاول اسمه calc.c وهو الموجود بالشكل رقم ١٢-٥ والملف الثاني اسمه tools.c وموجود بالشكل رقم ١٣-٥

```
0:  /*Program Name CALC.C*/
1:  #include "tools.c"
2:  #include <stdio.h>
3:  int sum(int a,int b);
4:  int mul(int x,int y);
5:  main ( )
6:     {
7:        int no1=5,no2=10;
8:        printf("\n\n no1+no2=%d",sum(no1,no2));
9:        printf("\n\n no1*no2=%d",mul(no1,no2));
10:    }
```

شكل رقم ١٢-٥ ملف البرنامج الرئيسي calc.c

وتلاحظ في بداية البرنامج الأول الموجود في الشكل رقم ١٢-٥ التوجيه "tools.c" #include ومعناه اربط الملف tools.c مع هذا الملف في مرحلة الترجمة وبالتالي تصبح الدوال الموجودة بالملف tools.c كأنها بالملف calc.c

```
0:  /*Program Name TOOLS.C*/
1:  int sum  (int a,int b)
2:     {
3:        int yt;
4:        yt=a+b;
5:        return yt;
6:     }
```

```
7:  int mul (int x,int y)
8:      {
9:        int k;
10        k=x*y;
11        return k;
21      }
```

شكل رقم ١٣-٥ ملف الدوال tools.c

الطريقة الثانية وهى الطريقة المتبعة فى البرامج الكبيرة وهى انشاء مشروع(project) هذا المشروع عبارة عن برنامج رئيسى يشتمل على دالة رئيسية ( )mainوبرامج فرعية تشتمل فقط على دوال ولاتحتوى على دالة () main ويتم ذلك من خلال بيئة كتابة برامج لغة C ولتطبيق ذلك على الملفين السابقينCALC.C,TOOLS.C.

اتبع الخطوات الاتية

١. افتح ملف البرنامج calc.c (راجع شكل ١٢-٥)
٢. احذف السطر رقم ١ من البرنامج ثم احفظ الملف
٣. اضغط مفتاحى alt+p تظهر قائمة project
٤. من قائمة project اختر new تظهر نافذة صغيرة تكتب فيها اسم المشروع
٥. اكتب اسم لهذا المشروع وليكن main ثم اضغط مفتاح الادخال
٦. اضغط مفتاح insert تظهر قائمة بها اسماء الملفات الموجودة عندك
٧. اختر الملف tools.c ثم الملف calc.c
٨. اضغط مفتاح Esc بعد الانتهاء من اختيار ملف المشروع
٩. اضغط مفتاحى Ctrl+F9 لتنفيذ المشروع

وبهذ تحصل على ملف تنفيذى باسم المشروع وهو main

مثال

نختم هذا الفصل ببرنامج متكامل يجمع معظم المفاهيم التى شرحناها حتى الآن (انظر شكل رقم ١٤-٥) وهذا البرنامج يقوم بطباعة قائمة اختيارات على الشاشة عبارة عن عمليات الجمع والطرح والقسمة والضرب ويطلب من المستخدم تحديد اختيار وعندما يقوم المستخدم بتحديد اختيار يقوم البرنامج بتنفيذ هذا الاختيار.

اكتب البرنامج أو فتحه من القرص المصاحب للكتاب ثم نفذه وشاهد نتيجة التنفيذ.

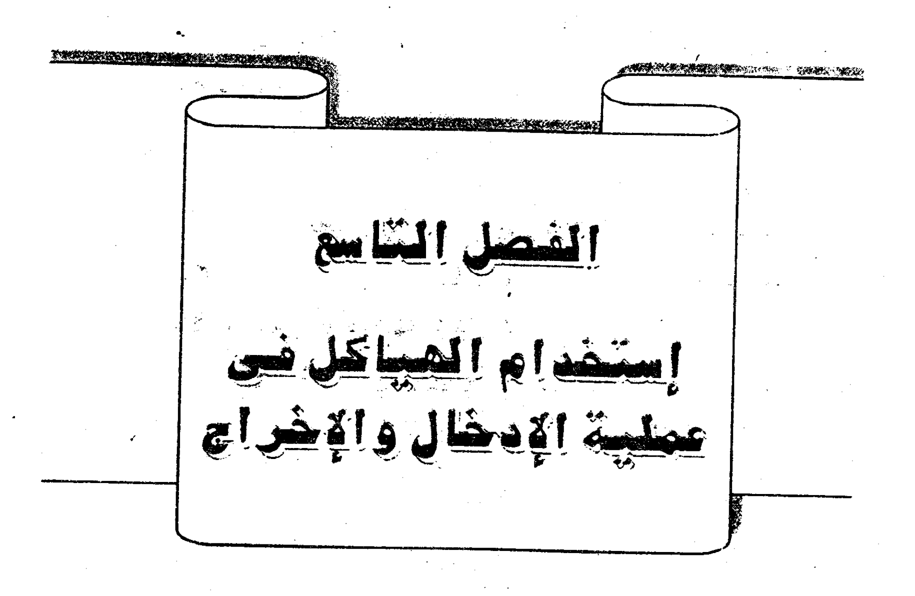
```
/*CS5_14.C
#include <stdio.h>
#include <conio.h>
void add(void);
void sub(void);
void mul(void);
void div(void);
main ()
   {
      char ch='0';
      while (ch!='5')
        {
   /* draw menu */
   printf ("\n*****************************************************\n");
   printf (" *************** main menu ******************** ****\n");
  printf ("\n (1)addation...");
  printf ("\n (2)sub..........");
  printf ("\n (3)mul..........");
  printf ("\n (4)Div..........");
  printf ("\n (5)exit.........");
  printf ("\n\n Enter selection :");
  ch=getch();
```

```
switch (ch)
    {
      case '1':
        add();
        break;
      case '2':
        sub();
        break;
      case '3':
        mul();
        break;
      case '4':
        div();
        break;
      case '5':
        ch='5';
        break;
      default:
        printf ("\n unknowan operator");
    }
  }
}

void add();
  {
    float no1,no2;
    char op;
    clrscr();
    printf ("\n enter no1,op,no2");
    scanf ("%f%c%f",&no1,&op,&no2);
    printf ("\n sum=%f",no1+no2);
```

```
void sub()
    {
      float no1,no2;
      char op;
      clrscr();
      printf ("\n enter no1,op,no2");
      scanf ("%f%c%f",&no1,&op,&no2);
      printf ("\n sub= %f",no1-no2);
    }
void mul()
    {
      float no1,no2;
      char op;
      clrscr();
      printf ("\n enter no1,op,no2");
      scanf ("%f%c%f",&no1,&op,&no2);
      printf ("\n mul= %f",no1*no2);
    }
void div()
    {
      float no1,no2;
      char op;
      clrscr();
      printf ("\n enter no1,op,no2");
      scanf ("%f%c%f",&no1,&op,&no2);
      printf ("\n div= %f",no1/no2);
    }
```

شكل رقم ١٤-د برنامج قائمة اختيارات يستخدم الدوال



الفصل التاسع

إستخدام الهياكل فى عملية الإدخال والإخراج

الفصل التاسع
إستخدام الهياكل فى عملية الإدخال والإخراج

استخدام Structure فى عمليات الإخراج:

ستقوم أولاً بفحص كيفية الحصول على structure كنتيجة من function . وفيما الكود الخاص بهذا الموضوع، متبوعًا بتوضيح للأوامر الموجودة فيه :

```
struct planet Gather(void)
{
   struct planet temp;

   cout << "ENTER THE PLANET NAME ";
   cin >> temp.name;
   cout << "ENTER THE PLANET DISTANCE ";
   cin >> temp.howfar;
   return(temp);
}
```

- يوضح function header أنه سيتم الحصول على structure اسمه struct planet كنتيجة للبرنامج الرئيسى فى ( ) Gather، ولن يكون هناك أية عناصر إدخال مطلوبة.
- يتم تعريف المتغير الداخلى بتحديد اسمه temp، ونوعه struct planet داخل هذه الـ function.
- ويقوم المستخدم فى سطرى الكود التاليين بإدخال اسم الكوكب، ثم بعده عن الشمس فى سطرى الكود التاليين لذلك أيضاً.
- ويتم تحديد البيانات الخاصة بالـ structure الداخلى فى المتغير temp، وتعرض محتويات temp كنتيجة للبرنامج الرئيسى.
- وأخيرًا يتم تحديد نتيجة Gather داخل البرنامج الرئيسى باستخدام المتغير third، الذى يمثل البيانات الخاصة بكوكب الأرض لأننا مازلنا نستخدم مثال النظام الشمسى.

```
third = Gather();
```

الحصول على النتيجة:

تعرض الخطوة التالية لتفاصيل ما تم جمعه من بيانات فى الكود السابق. ويكون من خلال ( ) Show، والتى يتم كتابتها على هذا النحو :

```
void Show(struct planet p)
{
   cout << endl << p.name << TAB << TAB;
   cout << p.howfar << endl;
}
```

• يتم نسخ محتويات المتغير third من البرنامج الرئيسى إلى المتغير P، الذى يعتبر متغير داخلى فى ( ) Show. وليس ثمة اختلاف فى ذلك مع ما ذكرناه عن المتغيرات البسيطة سابقاً. فمن خلال سطر الكود التالى، تلاحظ أنه لا يوجد أى اختلاف سوى أنه يتم استخدام structure هنا بدلاً من متغيرات . ويمكنك فقط تقرير ذلك بفحص prototype:

```
Show(third);
```

• يتم استخدام الأمر cout مرتين لعرض محتويات الحقل name ثم الحقل howfar على الشاشة.

ويعتبر كل هذا مجرد تكرار لما سبق شرحه فيما عدا أنك تستخدم الآن structure بدلاً من المتغيرات البسيطة.

طبيعة Structure المركبة

على الرغم من أن طبيعة structure طبيعة مركبة، حيث أنه يتكون من العديد من أنواع البيانات البسيطة، فإنه يمكن معالجته بنفس الطريقة التى تتم بها معالجة أنواع البيانات البسيطة مثل integer أو float، ويمكن الحصول عليه كنتيجة من function أيضاً.

مثال تطبيقى على استخدامات structures مع functions:

يقدم مثال (١٥-١) نموذج البرنامج الكامل لذلك. حاول تشغيله بإدخال البيانات الخاصة بكوكب الأرض. وستعرض الشاشة مايلى:

```
ENTER THE PLANET NAME : Earth
ENTER THE PLANET DISTANCE : 150
```

سيتم عرض البيانات المدخلة فى شكل مخرجات على الشاشة.

مثال (١٥-١): استعراض functions و structures البرنامج

```
#include <iostream.h>

#define LENGTH 20
#define TAB '\t'

struct planet {
              char name[LENGTH];
              int howfar;
       };

struct planet Gather(void);         // structure as output
void Show(struct planet p);         // structure as input

main()
{
  struct planet third;

  third = Gather();          // get details for earth
  Show(third);               // display the details
  return(0);
}

struct planet Gather(void)
{
  struct planet temp;

  cout << "ENTER THE PLANET NAME ";
  cin >> temp.name;
  cout << "ENTER THE PLANET DISTANCE ";
  cin >> temp.howfar;
  return(temp);
}

void Show(struct planet p)
{
  cout << endl << p.name << TAB << TAB;
  cout << p.howfar << endl;
}
```

إنشاء Array من مجموعة Structures:

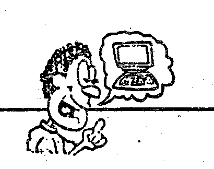
والآن سنحاول تكبير البرنامج ليشتمل على أكثر من structure واحد من خلال استخدام array تشتمل على مجموعة من structures .

ويتم فيه استخدام ( ) Gather أربع مرات لجمع بيانات الكواكب الأربعة الأولى . مع ملاحظة أن نتيجة Gather عبارة عن structure واحد ، لذلك يجب استخدامها أربع مرات لتحقيق هذه المهمة . وتذكر أنه يتم كتابة function مرة واحدة ولكن يمكن استخدامها مرات عديدة . ولقد سبق مناقشة هذا الموضوع فى الفصل الخامس . ويتم تخزين المخرجات الناتجة داخل متغير عبارة عن array تشتمل على مجموعة من structures . ويقوم كود function بكتابة كل المدخلات بحروف استهلالية كبيرة . ولقد ذكر هذا كثيراً من قبل ، ولكن يمكن توضيحه مرة ثانية فيمايلي :

```
for (x = 0; x < strlen(temp.name); x++)
  {
    temp.name[x] = toupper(temp.name[x]);
  }
```

تقوم جملة loop الأولى بفحص العنصر الأول من character array ، مع ملاحظة أن العنوان الخاص بهذا العنصر هو [0] . وتتأكد ( ) toupper من الشكل الذى تم به كتابة الحرف المدخل . فإذا كان بالفعل حرفاً استهلالياً كبيراً ، تركته function بدون تغيير . وإذا لم يكن كذلك ، قامت بتحويله . وتتم كتابة الحرف الناتج فوق العنصر الأول فى array . وبذلك يكون لديك حرف استهلالى كبير فى هذا الموقع .

وفى جملة loop الثانية ، تكون العملية مطابقة فيما عدا أنه يتم فحص العنصر الثانى ،



القيمة الإيجابية

تحتوى ( ) Gather الموجودة فى مثال (١٥-٢) على التعريف unsigned int x. ويسمى هذا التعريف modifier (أداة تحديد) ويعنى أن القيمة x التى من نوع integer مقصورة فقط على القيم الإيجابية من دون القيم السلبية. وستحتاج لهذا التعريف لأن strlen ( ) تنتج قيم إيجابية فقط، ولن تحصل منها على strings ذات أطوال سلبية. وعلى الرغم من أنه بدون هذا التعريف سيستمر البرنامج فى التشغيل، فإن compiler سيظهر تحذيراً. لذلك يجب استخدامه.

التى يعتبر العنوان الخاص به [1]. وتستمر العملية حتى نهاية string. وتقوم ( ) strlen الموجودة فى string.h library بتحديد عدد الأحرف التى يتكون منها والذى هو فى الواقع طول string. وباستخدام هذه الـ function يمكنك معرفة عدد المرات التى ستتكرر بها loop لأى string.

تسمح( ) Show بإدخال array تشتمل على مجموعة من structures إليها. ستخدم هنا أسلوب pass by reference لأن المدخلات فى شكل array. وتستخدم ملة for loop لاستعراض array وعرض البيانات منسقة على الشاشة.

وفيما يلى يتضح كيف أن البيانات المدخلة المكتوبة بأحرف صغيرة يتم عرضها بأحرف استهلالية كبيرة. وذلك لأن الكود يقوم بعملية تحويل الأحرف الصغيرة إلى أحرف استهلالية كبيرة.

```
ENTER THE PLANET NAME : Mercury
ENTER THE PLANET DISTANCE : 58
ENTER THE PLANET NAME : Venus
ENTER THE PLANET DISTANCE : 108
ENTER THE PLANET NAME : Earth
ENTER THE PLANET DISTANCE : 150
ENTER THE PLANET NAME : Mars
ENTER THE PLANET DISTANCE : 208

MERCURY    58
VENUS     108
EARTH     150
MARS      208
```

وإليك نموذج البرنامج المسؤول عن ذلك فى مثال (١٥-٢). حاول تنفيذه وقم بإدخال البيانات الخاصة بالأربع كواكب الأولى فى النظام الشمسى.

مثال (١٥-٢): عرض البرنامج

```
#include <iostream.h>
#include <string.h>
#include <ctype.h>

#define LENGTH 20
#define MAX 4
```

```
#define TAB '\t'

struct planet    {
            char name[LENGTH];
            int howfar;
        };

struct planet Gather(void);
void Show(struct planet p[MAX]);

main()
{
  struct planet system[MAX];
  int index;

  for (index = 0; index < MAX; index++)
  {
    system[index] = Gather();
  }
  cout << endl;
  Show(system);
  return(0);
}

struct planet Gather(void)
{
  struct planet temp;
  unsigned int x;

  cout << endl;
  cout << "ENTER THE PLANET NAME " ;
  cin >> temp.name;
  // Convert to uppercase
  for (x = 0; x < strlen(temp.name); x++)
  {
    temp.name[x] = toupper(temp.name[x]);
  }
  cout << "ENTER THE PLANET DISTANCE ";
```

```
cin >> temp.howfar;
   return(temp);
}

void Show(struct planet p[MAX])
{
   int loop;

   for (loop = 0; loop < MAX; loop++)
   {
      cout << p[loop].name << TAB << TAB;
      cout << p[loop].howfar << endl;
   }
   cout << endl;
}
```

ويعتبر كل هذا فى الواقع مجرد تكرار وتوضيح للمفاهيم المهمة التى سبق وشرحناها .

الاستمرار فى موضوع Functions:

لقد تم شرح جانب كبير من موضوع functions، ولكن مازالت هناك بعض النقاط الهامة التى سيتم توضيحها فيما يلى .

استخدام أكثر من عنصر إدخال واحد فى Function:

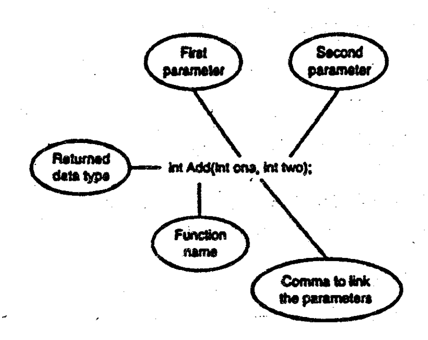
لم يخرج الأمر فى أمثلة functions السابقة عن أمرين اثنين :

إما أن Fuction لا تحتوى على عناصر إدخال على الإطلاق (void)، وإما أن تشتمل على عنصر إدخال واحد .

والآن، نحن بصدد العرض لحالة أخرى من استخدامات fuction تكون Fuction فيها محتوية على أكثر من عنصر إدخال واحد .

وسيتضح من البرنامج المعروض فى مثال (١٥-٣) ما هى الكيفية التى يمكن بها وضع عنصرين من عناصر الإدخال داخل function واحدة. ولكن دعنا نوضح الآن - قبل الدخول فى تفاصيل هذا البرنامج - fuction جديدة يعرض لها هذا النموذج وهى Add ( ) .

تحتوى ( ) Add على عنصرين اثنين من عناصر الإدخال يتم وضعهما داخل قوسين في حين يفصل بينهما (,). ويكون ناتج تنفيذ function في هذه الحالة قيمة معينة في مدى integer. وذلك بالفعل ما يوضحه الشكل التالى :



كيفية كتابة كود function تحتوى على أكثر من مدخل واحد

لعلك لاحظت معى الآن أن تعريف ( ) Add فى هذه الحالة قد اشتمل على متغيرين داخليين : One وTwo لتحديد عناصر عناصر الإدخال، فى حين استخدم المتغير (rec) لعرض عنصر الإخراج. على ذلك، يكون نموذج الكود الكامل لذلك على هذا النحو :

```
int Add(int one, int two)
{
    int res;

    res = one + two;
    return(res);
}
```

استخدام Functions تحتوى على عنصرى إدخال:

يوضح مثال (١٥-٣) خطوات البرنامج الكامل لاستخدام function تحتوى على أكثر من عنصر إدخال، وسأشرح لها الآن باختصار فيما يلى :

- يتم إستدعاء ( ) Gather مرتين داخل البرنامج الرئيسى ثم تحدد القيم باستخدام المتغيرين first و second.

• ويتم بعد ذلك إدخال عنصرى الإدخال first وsecond إلى ( ) Add ، وتعرض النتيجة فى المتغير result .

• وأخيراً يتم عرض إجمالى القيم فى المتغيرين first وsecond على الشاشة باستخدام ( ) Show .

وسيكون نموذج مخرجات هذا البرنامج على النحو التالى بعد إضافة القيمة 5 إلى قيمة 4 والحصول على الناتج 9:

```
ENTER A NUMBER 4
ENTER A NUMBER 5

THE TOTAL IS 9
```



كتابة Function مرة واحدة

ولعله من أهم أسباب تجزئة الكتابة فى البرنامج أن يقل حجم الكود المستخدم بالشكل الذى لا يحول دون سرعة التنفيذ. وسنرى فى مثال (١٥-٣) كيف يتم كتابة Function مرة واحدة فى حين يمكن استخدامها أكثر من مرة فى البرنامج. ويؤدى ذلك إلى توفير الكثير من الوقت عما لو تم كتابة Function فى كل مرة يتم استدعاءها فيها.

مثال (١٥-٢): نموذج البرنامج تحتوى على Function فيه على أكثر من عنصر إدخال واحد

```
#include <iostream.h>

int Gather(void);
int Add(int one, int two);
void Show(int total);

main()
{
   int first;
   int second;
   int result;
```

```
first = Gather();
   second = Gather();
   result = Add(first,second);
   Show(result);
   return(0);
}

int Gather(void)
{
      int x;

      cout << "ENTER A NUMBER ";
      cin >> x;
      return(x);
}

int Add(int one, int two)
{
   int res;

   res = one + two;
   return(res);
}

void Show(int total)
{
   cout << endl << "THE TOTAL IS  " << total << endl;
}
```

تحذير

يمكن فى المثال السابق استخدام عدد لانهائى من المدخلات، ولكن لا يفضل ذلك حتى لا تكون قراءة الكود صعبة على المستخدم.

ليس بالضرورة أن تكون المدخلات المستخدمة من نفس نوع البيانات . فيمكنك استخدام خليط من أنواع البيانات المختلفة كعناصر إدخال . وإليك هذا النموذج من prototype التالى ، الذى يشتمل على ثلاث مدخلات ، المدخل الأول من نوع character والثانى من نوع integer ، والثالث من نوع float :

```
int Example(int alpha, char beta, float gamma);
```

وهكذا ، فقد تم وضع ثلاث عناصر من عناصر الإدخال داخل Example ( ) ، ولكن مع مراعاة ترتيب هذه المدخلات إليها . ويمكنك فهم ذلك أكثر من الكود القادم الذى ستلاحظ فيه أن prototype قد حدد أن المدخل الأول يجب أن يكون char ، والثانى integer ، والثالث float . ويجب الالتزام بهذا الترتيب لضمان خروج البرنامج على النحو اللائق . وستكون المخرجات على النحو التالى :

```
42
a
9.275
```

مثال (٤-١٥) : مثال لعناصر إدخال تتكون من أنواع مختلفة من البيانات

```
#include <iostream.h>

void Example(int alpha, char beta, double gamma);

main()
{
    int a = 42;
    char b = 'a';
    double g = 9.275;

    Example(a,b,g);
    return(0);
}

void Example(int alpha, char beta, double gamma)
{
   cout << alpha << endl;
   cout << beta << endl;
   cout << gamma << endl;
}
```

استخدام الصيغ المختصرة الخاصة بلغة ++C:

لقد وعدت فى بداية الكتاب أننى سأقدم مجموعة من الصيغ المختصرة الخاصة بلغة ++C عند الحاجة لذلك. وفيما يلى عرض لإحدى هذه الاختصارات. فيقوم البرنامج فى مثال (١٥-٥) بعرض إحدى الطرق المختصرة لكتابة الكود كتلك الموجودة فى مثال (١٥-٤) ولكن توجد بعض الاختلافات الهامة التى سأوضحها الآن.



تبديل المتغيرات

حاول تغيير ترتيب المتغيرات a و ا فى ( ) Example السابق عرضه فى مثال (١٥-٤) ولاحظ الناتج ستجد أن برنامجك لن يعمل بطريقة لائقة.

مثال (١٥-٥): الصيغة المختصرة لتنفيذ عملية الإدخال

```
#include <iostream.h>

int Gather(void);
int Add(int one, int two);
void Show(int total);

main()
{
    Show(Add(Gather(),Gather()));
    return(0);
}

int Gather(void)
{
     int x;

     cout << "ENTER A NUMBER ";
     cin >> x;
     return(x);
}

int Add(int one, int two)
{
```

```
    // another piece of shorthand
        return(one + two);
}

void Show(int total)
{
    cout << endl << "THE TOTAL IS  " << total << endl;
}
```

وهكذا يكمن الاختلاف الأساسى فى البرنامج الرئيسى.

```
main()
{
    Show(Add(Gather(),Gather()));
    return(0);
}
```

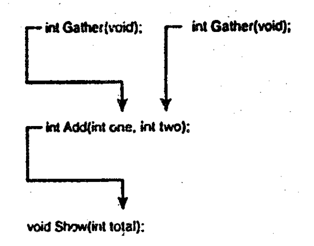
لاحظ أولاً أنه لا توجد أية متغيرات فى function التنفيذ. وكان ذلك على النحو التالى:

```
Show(Add(Gather(),Gather()));
```

يحل هذا السطر محل الأربعة أسطر التالية فى البرنامج الأصلى

```
first = Gather();
second = Gather();
result = Add(first,second);
Show(result);
```

والآن، انظر إلى الشكل القادم الذى يوضح كيفية اشتمال ( ) Add على عنصرين من عناصر الإدخال من نوع integer. ولاحظ كيف قامت ( ) Gather بجمع قيمة هذين العنصرين، فكان الناتج أيضاً فى ( ) Add قيمة معينة فى مدى integer، أظهرتها ( ) show عند تنفيذها كنتيجة على الشاشة. ونستخلص من ذلك كله الحكم بإنسجام جميع functions البرنامج فى هذه الحالة بالشكل الذى لم يكن من الضرورى معه اللجوء إلى متغيرات وسيطة لأن عناصر الإخراج فى function معينة يمكن أن تكون عناصر الإدخال والإخراج فى functions أخرى.



انسجام functions البرنامج بالشكل الذى
لا يلزم معه استخدام متغيرات وسيطة

ويكمن الاختلاف الآخر فى ( ) Add فى أنها تشتمل على سطر واحد فقط من الكود بدلاً من السطرين الأصليين ويقوم المتغيران الداخليان one و two اللذان تم تحديدهما فى بداية البرنامج الرئيسى بتقديم نتيجة واحدة عند جمعهما معًا الأمر الذى يوفر عليك تعريف المتغير الداخلى، بجمع الرقمين معًا ثم تقديم ذلك الرقم فى شكل نتيجة.

```
return(one + two);
```



نظرة عامة حول مفهوم Function Crunching

تعرف طريقة استخدام function كمدخل لـ function أخرى باسم Crunching function وهى تقنية يستخدمها المبرمجون على نطاق واسع. وبالإضافة إلى حدها من سول البرنامج، فإنها تزيد أيضًا من سرعة تنفيذه الأمر الذى لا تتضح أهميته إلا مع البرامج الكبيرة. ومع ذلك فهناك جانب سلبى لهذه الطريقة يكمن فى صعوبة قراءة الكود. ولذلك أحاول دائمًا تجنب هذه التقنية فى الدراسة، ولكن عند كتابة برامج حقيقية، سيكون الأمر مختلفاً.

نتيجة ( ) main:

ليس البرنامج الرئيسى الذى يقوم باستدعاء جميع functions البرنامج الأخرى فى الواقع سوى function يتم استدعاءها من نظام التشغيل. وتشتمل عادة على header التالى:

```
main()
```

تذكر أنك فى كل ما سبق ذكره عنfunctions قد قمت بتحديد نوع البيانات فى النتيجة التى ستحصل عليها، حتى ولو كانت void. وفى الواقع، فإن نوع النتيجة الافتراضى لأى function يكون قيمته فى مدى integer، لذلك لن تحتاج لتحديدها. ولكن لكى تكون برمجتك سليمة، فلابد من أن تحدد نتيجة ( ) main على أنها قيمة فى مدى integer كالتالى:

```
int main()
```

ومع افتراض أن البرنامج الرئيسي قد أظهر نتيجة قيمة فى مدى integer، فإن ذلك يعنى انتهاء ( )main التى تنتهى بدورها functions البرنامج الأخرى ويستخدم السطر التالى:

```
return(0);
```

وهكذا يتضح أن القيمةمن نوع integer، ولعلك عرفت الآن سبب وجودها هنا ومن أين جاءت فى الأصل.

وبدلاً من ذكر (0) return فى نهاية كل برنامج، يمكنك تحديد ( )main فى صورة void return وفى هذه الحالة لن تحتاج لتقديم أية نتيجة. وتعتبر هذه طريقة شهيرة أيضًا فى تنفيذ المهام المختلفة. وإليك البرنامج التالى فى مثال (١٥-٦) الذى يوضح كيفية استخدام void return.

مثال (١٥-٦): استخدام البرنامج الرئيسى لـ Void Return

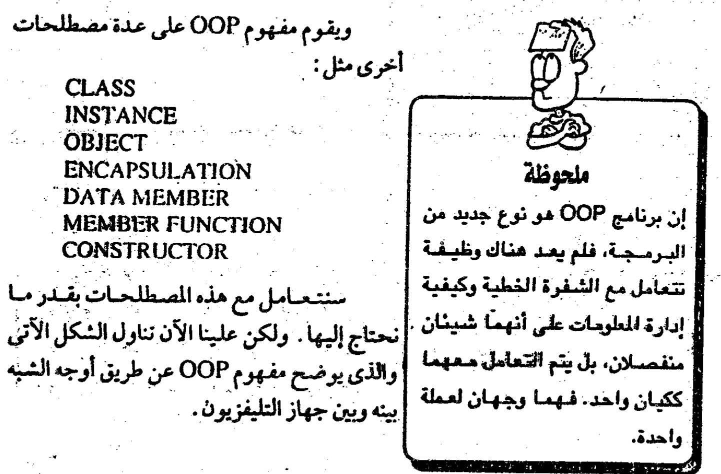
```
#include <iostream.h>

void main()
{
   cout << "Look, no return needed!" << endl;
}
```


الفصل العاشر
مقدمة فى البرمجة
شيئية التوجه

مميزات برنامج OOP:

فى الماضى كان المبرمجون يكتبون الشفرة التى تدير المعلومات، ولكنهم كانوا يتعاملون مع المعلومات والشفرة على أن كلاً منهما شئ منفصل تمامًا عن الآخر. إلا أن برنامج OOP، من ناحية أخرى، يتعامل مع الشفرة والمعلومات على أنهما كيان واحد يعرف باسم class. سنسوق مثالاً بسيطاً يوضح مفهوم class واستخدامه، وعندما تدرك هذا المفهوم ستكون قادراً على تطوير برامج object-oriented الأكبر والأكثر تعقيداً. تعرف برامج object-oriented باسم OOP أو OO.



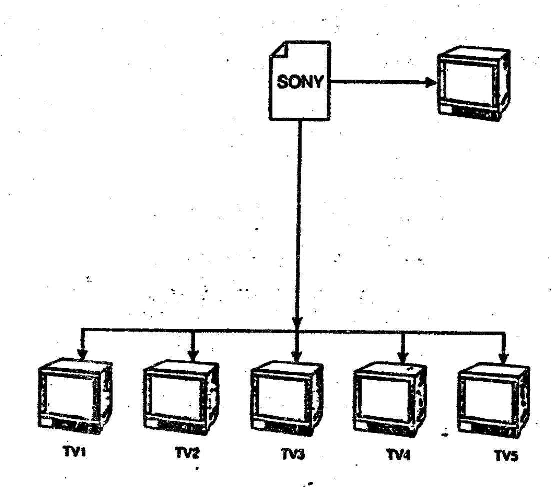
ويقوم مفهوم OOP على عدة مصطلحات أخرى مثل:

CLASS
INSTANCE
OBJECT
ENCAPSULATION
DATA MEMBER
MEMBER FUNCTION
CONSTRUCTOR

ملحوظة

إن برنامج OOP هو نوع جديد من البرمجة، فلم يعد هناك وظيفة تتعامل مع الشفرة الخطية وكيفية إدارة المعلومات على أنهما شيئان منفصلان، بل يتم التعامل معهما ككيان واحد. فهما وجهان لعملة واحدة.

سنتعامل مع هذه المصطلحات بقدر ما نحتاج إليها. ولكن علينا الآن تناول الشكل الآتى والذى يوضح مفهوم OOP عن طريق أوجه الشبه بينه وبين جهاز التليفزيون.



شكل يوضح العلاقة بين تركيب جهاز تليفزيون من دائرة كهربية وتركيب عدة أجهزة من نفس هذه الدائرة الكهربية.

أسلوب Command -line فى البرمجة:

تحدثنا فيما سبق عن كيفية وضع عناصر الإدخال فى functions معينة فيما بين الأقواس، وعرفنا كيف يطبق ذلك على ( ) main أيضاً. وقد كانت هذه هى الطريقة الشائع إتباعها فى البرمجة بأسلوب command line، والتى كان يستخدم فيها نظم التشغيل Dos أو UNIX. ومازالت تستخدم أيضاً فى بيئات تشغيل أخرى مثل: Windows.

وسأضمن الفقرة التالية شرحاً موجزاً للبرمجة بهذا الأسلوب.

تختلف ( ) main عن باقى functions البرنامج العادية حيث لابد من الالتزام بترتيب معين لإدخال عناصر البيانات إليها. ولا يجوز أبداً الخروج عن هذا النسق أو مخالفته. وإليك فيما يلى عرضاً سريعاً لهذا التنسيق:

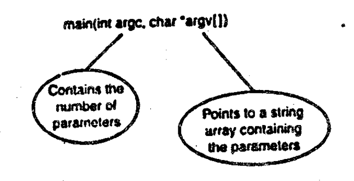
افترض أنه لديك برنامج يطبق أسلوب command - line السابق، وكان اسمه AVERAGE، يمكنك الحصول على متوسط القيمة العددية لعنصرى الإدخال على النحو التالى:

AVERAGE 8 4



التفاضل بين لغات البرمجة المختلفة

تتعدد أساليب البرمجة التى تتيحها لغات C و++C بالشكل الذى يسمح باستخدام وكتابة أكثر من كود واحدة لتنفيذ الكثير من المهام. وذلك فضلاً عما تشتمل عليه من صيغ مختصرة تزيد من سرعة التنفيذ والاستخدام. ولعل هذا كله ما أدى إلى تصدر لغة ++C لجميع اللغات الأخرى. حتى أنه بمقارنتها مع لغة أخرى مثل Visual Basic لا يكون من المهم فى شئ المجهود الكبير المبذول فى تعلمها فى مقابل ما تتيحه من مرونة الاستخدام.



كيفية كتابة main بأسلوب Command -line

والآن سيتم تحديد القيمة المبدئية فى argc على أنها3، ويعنى ذلك أن argv* عبارة عن array تحتوى على ثلاثة عناصر من نوع string . وستتمثل محتويات العناصر داخل array [ ] argv* فيما يلى :

```
*argv[0]     "AVERAGE"

*argv[1]     "8"
*argv[2]     "4"
```

وعلى ذلك يمكنك للوصــول إلى اسم برنامج Command -line أن تستــدعى العنصر الأول من array والذى يحمل العنوان صفر [0] :

```
cout << argv[0];
```

أما الوصول إلى القيمتين العدديتين، فيكون من خلال العنصرين الثانى والثالث فى array، وذلك باستخدام العنوانين[1] و [2] على التوالى :

```
cout << argv[1];
cout << argv[2];
```

ولاحظ أن القــيم العــددية التى تمر إلى main هى فى الواقع arrays من نوع character ولذلك يمكن اعتبارها أيضاً strings . وسنرى الآن كيفية استرجاع هذه القيم حيث يقوم البرنامج فى مثال (١٥-٧) بجمع القيمتين معاً ثم عرض النتيجة على الشاشة .

نظم التشغيل المختلفة ونشأة أسلوب command - line فى البرمجة

كان من الضرورى فى نظم التشغيل القديمة فى بيئة DOS وUNIX أن يكتب الأمر متبوعاً ببعض البيانات – ثم تستدعى function معينة للتعامل مع هذه البيانات وتحليلــها وإظهــار النتيجــة بعد ذلك. ومن هنا جاءت فــكرة Command -line. وتميزت هذه النظم بأنها موجزة وملائمة وليست فى حاجة إلى حيز كبير فى الذاكرة لتنفيذ مهام البرنامج. ولعل ذلك هو السبب فى استمرار استخدام قاعدة UNIX الضخمة وكذلك نظام التشغيل Dos على الرغم من ظهور نظام التشغيل الأحدث المتمثل فى برامج Windows.

مثال (١٥-٧): نموذج لبرنامج Command -line

```
// argc is the number of parameters in argv
// argv is a pointer to the actual string parameters
// argv[0] points to the name of the program

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
    int first;
    int second;
    int third;

   // This is error checking
    if (argc < 3)
    {
       cout << "MUST BE MORE THAN ONE PARAMETER"
           << endl;
       exit(0);
    }
   // This is the calculation
     first = atoi(argv[1]);
     second = atoi(argv[2]);
     third = first + second;
  // Display program title
     cout << argv[0] << endl;
  // Display the result
     cout << "THE SUM OF " << first << " AND "
         << second << " IS " << third << endl;
     return(0);
}
```

تشغيل برنامج Command -line:

يتم تشغيل وحفظ وإجراء عملية Compilation لهذا البرنامج مثل باقى برامج لغة C++ تمامًا. ولكن ثمة اختلاف وحيد فى أنه يجب إدخال معلومات إضافية عند تشغيل البرنامج. وسأعرض الآن طريقتين لتنفيذ ذلك.

الطريقة الأولى:

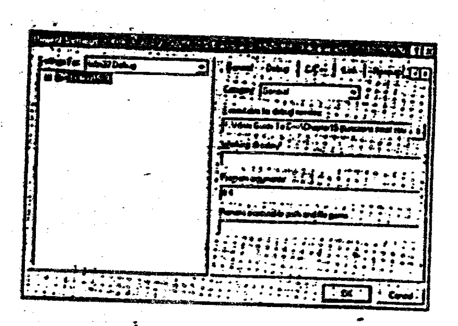
يتم إنتاج exe file فى جميع برامج التطبيق الخاصة بلغة ++C بعد تنفيذ عملية الربط فيما بينها. ولذلك، يمكن الاستفادة منها جميعها فى نظام التشغيل Dos. وإليك نموذج للبرنامج الذى ينفذ ذلك، والذى اختير اسم الملف فيه ليكون L15-7 حيث لا يجوز فى بيئة تشغيل Dos استخدام أسماء تزيد عن ثمانية أحرف لا توجد بينها مسافة. ويمكنك تنفيذ ذلك بنفسك بكتابة الكود التالى:

L15_7 8 4

وسيقوم DOS بتحميل البرنامج وتشغيله لإظهار الناتج 20 على الشاشة قبل ظهور رسالة الحاث DOS مرة أخرى. ويقف التنفيذ عند هذا الحد فى حالة صحة البرنامج. لكن إذا وجدت خطأ، يجب عليك إعادة تحميل البرنامج وتصحيح الأخطاء.

الطريقة الثانية:

بالاعتماد على إصدار لغة ++C الذى تستخدمه، ستجد داخل إحدى القوائم المنسدلة خياراً يسمح لك بإدخال بيانات. ففى لغة ++Visual C الخاصة بشركة Microsoft، يوجد هذا الخيار داخل Project Settings ، Debug ، Program Argument. افتح مربع هذا الخيار واكتب بيانات Command -line التى تريدها فى برنامجك، كما هو موضح فى الشكل التالى:



وضع عناصر الإدخال بأسلوب Command -line فى ++Microsoft Visual C

والآن قم بتشغيل البرنامج . وسيتم إدخال البيانات أثناء التشغيل .

```
Listing15_7.exe
THE SUM OF 8 AND 4 IS 12
```

لا تنس أن الطريقة الثانية تستخدم فقط لاختبار وتقييم البرنامج . فيتم إعداد برامج Command -line ليتم تشغيلها باستخدام نظام Command -line مثل DOS .

شرح البرنامج:

يتكون البرنامج من ثلاثة أجزاء . سأعرض كل جزء منها على حدة .

مفهوم Command line:

إن Command line هو السطر الذى يمكننا من إدخال بيانات إلى حيز الأداء فى البرنامج الرئيسى ويكون ذلك على النحو التالى :

```
main(int argc, char *argv[])
```

لقد أوضحت من قبل ما الذى يعنيه ذلك الكود . فتأكد أنك فهمت هذا المفهوم . وعليك أن تعلم أن أسماء العناصر argc و argv إجبارية ولا يمكنك استخدام أسماء غيرها على نحو ما كان يحدث فى functions البرنامج العادية .

موقع الخطأ:

كان الهدف من هذا البرنامج الحصول على حاصل جمع قيمتين عدديتين وعرضه على الشاشة . وقد لزم لذلك وجود ثلاث عناصر إدخال لاستدعاء البرنامج من سطر الأمر . وقد كان من الضرورى أولاً التأكد من صحة عدد عناصر الإدخال من خلال الشرط التالى :

```
if (argc < 3)
{
    cout << "MUST BE MORE THAN ONE PARAMETER");
    exit(0);
}
```

فإذا تم إدخال أى شئ أقل من الاسم والقيمتين، ستعرض رسالة تنبيه وتوضح ذلك، وسينتهى البرنامج . ويمكنك إدخال عدد أكثر من ثلاث مدخلات، ولكن سيتم تجاهل فارق الزيادة .

تخفى الأرقام:

ليست القيم العددية فى الواقع إلا srting مكونة من عدة أرقام، يجب تحويلها إ قيم حقيقية. ويستخدم فى ذلك functions عديدة موجودة فى ibrary stdlib.h وسيتم الآن استخدام atoi التي تحول ASCII characters إلى قيمة من نوع integer :

```
first = atoi(argv[1]);
second = atoi(argv[2]);
```

وعلى ذلك، تشتمل first و second الآن على قيم عددية يمكنك جمعها ببساط على النحو التالى:

```
third = first + second;
```

لقد قمت بكتابة عنوان البرنامج على الشاشة لتوضيح كيفية تنفيذ هذا. ولكن لا يعتبر ذلك إجباريًا:

```
cout << argv[0];
```

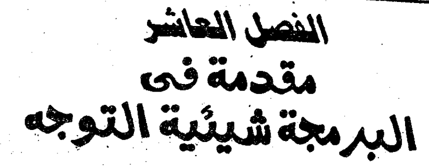
وأخيرًا تقوم بعرض النتيجة على الشاشة بالطريقة العادية وتنهى البرنامج بالحصول على القيمة التى من نوع integer.

```
cout << "THE SUM OF " << first << " AND "
        << second << " IS " << third << endl;
return(0);
```

أهم الدروس المستفادة:

- على الرغم من أن طبيعة structure مركبة حيث يتكون من أنواع عديدة من البيانات البسيطة، فيمكن الحصول عليه كنتيجة من function بنفس طريقة إنتاج أنواع البيانات البسيطة مثل integer أو float.
- على الرغم من طبيعته المركبة، يمكن إدخال structure إلى function بنفس طريقة إدخال المتغير البسيط.
- يمكنك الوصول إلى حقول structure داخل function باستخدام رمز النقطة (.). ويتم مثل هذا تماماً فى برنامج ( ) main.
- يتم إدخال arrays من نوع structure إلى functions بأسلوب pass by reference كما يحدث تماماً مع أية array أخرى.

- يمكن أن يكون للـ function مدخلات عديدة، يتم ربطها معاً باستخدام
- ليست هناك ضرورة لأن تكون جميع عناصر الإدخال فى unction البيانات. فيمكن أن تكون من أى نوع تريده.
- إذا كان للـ function عناصر إدخال متعددة من أنواع بيانات مختلفة، بالترتيب الصحيح للمدخلات ليتم تشغيل function بطريقة سليمة.
- main نفسها مجرد function خاصة، يمكن من خلالها تشغيل باق البرنامج.
- يمكن استخدام ( ) mai فى صورة Command-line function فى التى لا تعمل بنظام Windows مثل DOS أو UNIX.

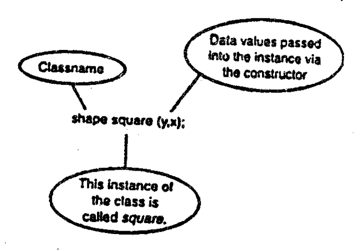


فمن الممكن أن نشبه class بشكل الدائرة الكهربية لجهاز التلفزيون Sony. إن شكل الدائرة الكهربية هو مجرد قطعة من الورق توضح كيفية تركيب جهاز التليفزيون، ولكنها ليست جهاز التليفزيون الفعلى، فعندما تم رسم الدائرة الكهربائية كمخطوط على الورق يوضح تركيب جهاز التليفزيون، كانت النتيجة هى جهاز تليفزيون فعلى بإمكانك لمسه واستخدامه، أى أنك صنعت نموذجاً لجهاز تليفزيون.

ومن مخطوط الدائرة الكهربية الواحدة يمكنك صنع العديد من أجهزة التليفزيون. فكل لحظة تصنع فيها نموذجاً لجهاز تليفزيون معتمداً على المخطوط ينتهى الأمر بحصولك على جهاز تليفزيون حقيقى آخر. وعلى الرغم من تماثل كل أجهزة التليفزيون إلا أن لكل منها كيانه المتفرد الذى يميزه عن الآخر. وهذه الفكرة تسرى أيضًا على عناصر برنامج الكمبيوتر. فكل نموذج (instance) من نفس class لابد وأن يكون له اسم مميز.

كيفية عرض مفهوم Encapsulation:

إن كل برامج OOP تبدأ من class. يعرف class بأنه تركيبة البيانات التى تحتوى على كل مايلزم لتخزين وإدارة البيانات. ففى الماضى، كان يتم تخزين البيانات داخل المتغيرات لكن فى OOP، فإن أى متغير مميز داخل class يطلق عليه data member وهذا لتمييزه عن المتغير العادى. أما وظائف إدارة البيانات فيطلق عليها member functions وهذا أيضًا لتمييزها عن الوظائف العادية. ولا يمكن إدارة data member إلا عن طريق member functions فالوظائف والأوامر الخارجية لا يمكنها إدخال data member، ولهذا السبب يرتبط كلاً من data members و member functions معًا فى كيان واحد يسمى class. فلا تأثير لأحدهما بدون الآخر وهذا هو ما يسمى encapsulation. الشكل الآتى يوضح مفهوم class، فكل ما فى الأقواس يتسم به encapsulated.



شكل يوضح تعريف class فى لغة C++



لغة ++C

إنك الآن تدخل عالم البرمجة المتخصصة. إن لغة OOP كانت تسمى لغة ++C وكانت تستخدم أمر الإخراج cout.



هل Class يشبه Structure؟

هذا صحيح ولكنهما ليسا نفس الشئ. فلا تخلط بينهما فهما مختلفان.

والآن عليك ممارسة هذا التدريب البسيط. قم بعمل برنامج يعتمد على OOP ويستطيع حساب مساحة المربع. إن كل ما سوف تقوم به هو عمل قالب ليرشدك إلى الخطوات التى تندرج تحت أسلوب برمجة OOP.

دليلك إلى برنامج OOP:

تشكل الخطوات الست القادمة قالباً لبناء أسلوب برنامج OOP. اتبع هذه الخطوات وحاول أن تدرك ما تقوم به من عمليات. عندما تقوم بكتابة البرنامج، استبدل data member و member functions لتحقيق النتائج المرجوة، عليك بالجمع والتصنيف فى كل مرحلة، فهذا يسهل عليك إلتقاط الأخطاء مبكراً، وهذا مهم عند دراسة OOP حيث أن الخطأ الواحد قد يؤدى إلى أخطاء مضاعفة. إن الخطوات من الأولى إلى الخامسة لا تعطى أية مخرجات على الشاشة، فلن ترى أية نتائج قبل الخطوة السادسة. فعليك بالصبر.

الخطوة الأولى فى أسلوب OOP "بناء class":

الخطوة الأولى هى إنشاء وبناء class اسمه shape. انظر إلى المثال (١٩-١). هذا المثال للإيضاح فقط ولا يقوم بأية مهام أخرى، فهو مجرد قالب.

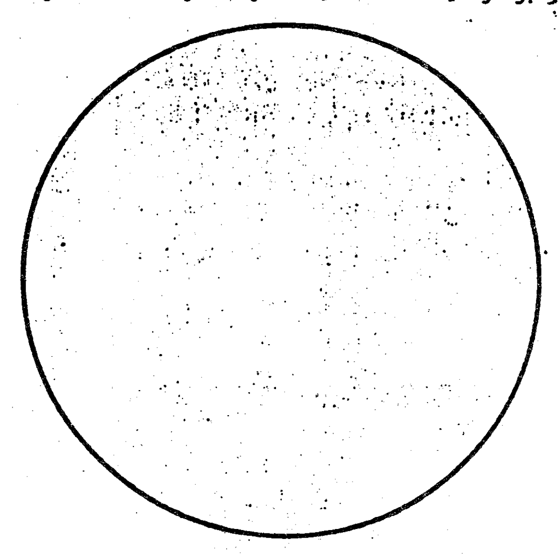
مثال (١٩-١): كيفية بناء class

```
#include <iostream.h>

class shape {
// AN EMPTY CLASS
        };

main()
{
   return(0);
}
```

ويمكن أن تتخيل class على أنه عالم أو دائرة تضم عدداً من العناصر تتبع class، ولكنه فى المثال السابق لا يضم أى شئ لأننا لم نحدد عناصر أو وظائف shape. لا تسمح الحدود الصلبة لهذه الدائرة بوصول أية مدخلات حيث لا توجد ثغرات فارغة داخل محيط الدائرة.



شكل يوضح مفهوم class فارغ

الخطوة الثانية فى أسلوب OOP:

إن الخطوة الثانية هى إعطاء Class مجموعة من data members، وهى عبارة عن البيانات التى تنتمى إلى class فقط. ويوضح مثال (١٩-٢) كيفية حدوث ذلك.



لا يزال Class تشبه Structure

إن class لديه ما يشبه المتغيرات فى أنواع البيانات المختلطة، فلا يوجد اختلافاً إلا فى اللغة فقط. نطلق على هذه المتغيرات اسم data members.

مثال (١٩-٢): إضافة Data Members إلى class

```
#include <iostream.h>

class shape    {
        // DECLARE DATA MEMBERS
        // WHICH ARE PRIVATE BY DEFAULT
        // BUT I HAVE INCLUDED THE WORD
        // TO SHOW THE CORRECT SYNTAX
      private:
        int length;
        int height;
        int area;
        };

main()
{
   return(0);
}
```

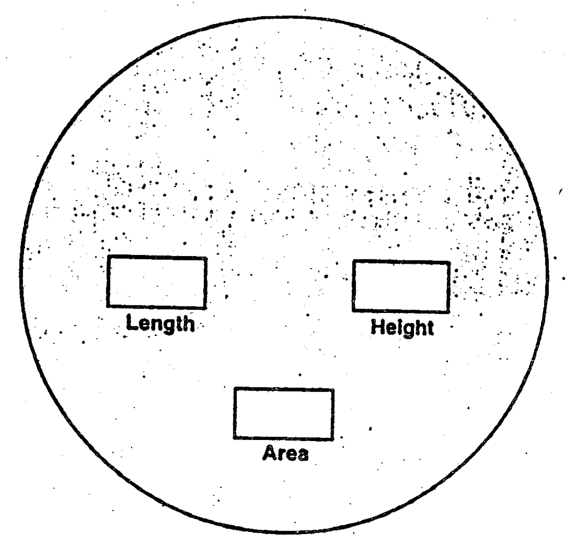


توضيح Private و Public

تستخدم كلمة Publicoop لوصف أي member في Class يمكن إدخاله من خارج Class.

تستخدم كلمة Private OOP لوصف أي member في class لا يمكن إدخاله من خارج class. إن Private members لا يمكن إدخالها إلا بواسطة member functions. هناك إتفاق في لغة C++ لإضفاء تعبير Private على كل data members، واعتبار ما دون ذلك برمجة خاطئة. إن أسلوب OOP يقوم على إخفاء البيانات داخل class، وإدارة هذه البيانات بواسطة member functions، وهذا هو ما نعنيه بمصطلح encapsulation. ولسوء الحظ لازال في الإمكان عمل class بلغة C++ يمكنه إدارة data member بدون الحاجة إلى member functions، وإن كان هذا يعد إرتداداً إلى الإصدارات الأولى من اللغة. إن كل members داخل class تعد افتراضاً private.

يوضح مثال (١٩-٢) كيف تم إضافة اثنين من data members يطلق عليهما اسم height و length إلى class، يعتبر كل منهما بصورة افتراضيةprivate، وهذا يعنى عدم إمكان إدخالهما عبر حدود class .



Length Height Area

شكل يوضح مفهوم class وبداخله ثلاث data members

الخطوة الثالثة فى أسلوب OOP:

Prototyping the Member Functions:

إن الخطوة التالية هى إنشاء member functions تستطيع إدارة data members .



Structure لا تحتوى على Functions

هذا صحيح، لكن من الأفضل أن نستبدل اسم functions باسم member functions .

مثال (١٩-٢): إضافة Member functions إلى Class

```
#include <iostream.h>

class shape    {
          // DECLARE DATA MEMBERS
          private:
             int length;
             int height;
             int area;

          public:

          // DECLARE MEMBER FUNCTIONS
          // NOTE THE EXPLICIT USE OF PUBLIC
          // THIS ALLOWS ACCESS FROM OUTSIDE
             void CalcArea(void);
             void ShowArea(void);
          };

main()
{
   return(0);
}
```

شكل يوضح class بداخله ثلاث data members واثنان من member functions

الشكل الآتى يوضح تأثيـر إضـافـة member functions إلى class . فـإذا كـان CalcArea أحد الأزرار فبمجرد الضغط عليه يقوم بإدخال الطول والإرتفاع وتشغيلهما قبل تخزين النتيجة كمساحة. سنعتبر member function المسمى ShowArea زر آخر يقوم بحمل مُحتويات المساحة كما ينقل البيانات عبر حدود class ويعرض النتائج على العالم الخارجي. لاحظ أن عبور الطول والإرتفاع باعتبار أنهما data members لحدود class لا يستغرق وقتاً طويلاً، فهما لا يزالان خارج class.



> **تعريف Constructor**
>
> إن constructor هو member function خاص. يمكنك دائمًا التعرف على constructor لأنه يحمل نفس اسم class الذى ينتمى إليه.

الخطوة الرابعة فى أسلوب OOP: وظيفة Constructor:

إن المشكلة التى لا تزال قائمة هى كيف يمكن وضع البيانات داخل الطول والإرتفاع فى المقام الأول؟ فنحن لا نستطيع عبور حدود الدائرة. ولكن هذا ممكن أن يحدث بواسطة نوع خاص من member function يسمى constructor. يضيف مثال (١٩-٤) يضيف constructor إلى class.

مثال (١٩-٤): إضافة Constructor إلى Class

```
#include <iostream.h>

class shape   {
        // DECLARE DATA MEMBERS
        private:
           int length;
           int height;
           int area;

        public:

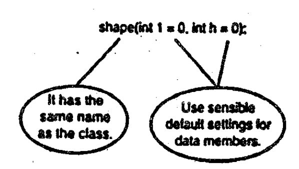
        // DECLARE MEMBER FUNCTIONS
           void CalcArea(void);
```

```
    void ShowArea(void);
    // CREATE A CONSTRUCTOR
       shape(int l = 0, int h = 0);
  };

main()
{
  return(0);
}
```

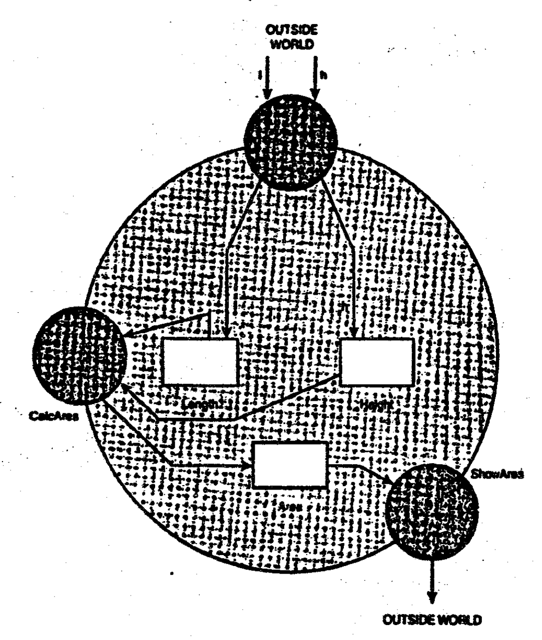
يوضح الشكل التالى تركيب constructor . يعد إضافة قيم افتراضية إلى constructor من أساليب البرمجة الجيدة، وإن كان ذلك ليس إجباريًا، وبالمثل تستطيع حذف أية إعدادات افتراضية . هذا سيتضح فى الشكل التالى :

```
shape(int l , int h );
```



شكل يوضح تركيب constructor بواسطة القيم الافتراضية

الشكل التالى يوضح أثر إضافة constructor إلى class . إن constructor يقوم جمع l وh public data من العالم الخارجى، ثم يقوم بنقل فيهما عبر حدود class يضع هذه القيم داخل private data members اسمها height وlength .



شكل يوضح مفهوم class وبه ثلاث data members واثنان من member functions بالإضافة إلى constructor

الخطوة الخامسة فى أسلوب OOP:

تعريف الكود الخاص بكل من Member Functions و Constructor:

إن الخطوة التالية هى تعريف member function وستقوم بذلك بطريقة مشابهة للكود الأساسى فى لغة ++C العادية، بالطبع سيكون هناك بعض الإختلافات الطفيفة .الاختلاف الأول بسيط : يتم تحديد كل member function مباشرة بعد تعريف class باستخدام الكود العادى للغة ++C، أما الإختلاف الثانى فيبدو غريبًا بعض الشئ لذلك سنوضحه فى المثال (١٩-٥).

لماذا نختار إعطاء قيم إفتراضية؟

إن هذا يشبه إلى حد بعيد تعريف وتحديد المتغيرات فى البرمجة التقليدية. تذكر أنه لم يكن تعريف المتغير كالاتى.

```
int number;
```

أو تعريفه وبدء تحميله كالآتى :

```
int number = 0;
```

إن إضافة قيم إفتراضية ليست إجبارية ولكنها من أساليب البرمجة الجيدة .

مثال (١٩-٥): إعطاء تعريف إلى Member Functions

```
#include <iostream.h>

class shape   {
        // DECLARE DATA MEMBERS
        private:
            int length;
            int height;
            int area;

        public:
        // DECLARE MEMBER FUNCTIONS
            void CalcArea(void);
            void ShowArea(void);
        // CREATE A CONSTRUCTOR .
            shape(int l = 0, int h = 0);
        };

// NOW DEFINE THE MEMBER FUNCTIONS

void shape::CalcArea(void)
```

```
{
   area = length * height;
}

void shape::ShowArea(void)
{
   cout << "THE AREA IS : " << area;
}

shape::shape(int l, int h)
{
   length = l;
   height = h;
}

main()
{
   return(0);
}
```

يكمن الإختلاف الكبير بين لغة ++C العادية وتعريفات وظائف أسلوب OOP فى أن الأخير يخصص class الذى ينتمى إليه التعريف بواسطة معامل تحليل المدى (scope resolution operator) ويكون على شكل (: :)، وهذا بدوره ينقل إلى compiler أن الوظيفة member function هى عضو (member) داخل class، ومن هنا جاءت التسمية member function . لاحظ أنه فى مضمون member functions توجد data members بشكل غير ظاهر وكأنها متغيرات عالمية . إن هذه ليست مشكلة فى أسلوب OOP بل هى ميزة فى الحقيقة. فأنت تجمع كل ماالديك من متغيرات فى class وتقوم بإدخالها بواسطة member functions ، وعن طريق استخدام scope resolution operator يستطيع compiler ترتيب أماكن هذه البيانات .

الخطوة السادسة فى أسلوب OOP:

بدء تنفيذ البرنامج بعد الإنتقال من Class إلى Object:

أخيراً وبعد كل هذه الخطوات، سيقوم البرنامج بتنفيذ شئ. انظر إلى مثال (١٩-٦)، فلأول مرة يظهر لنا كود فى () main، وإذا رجعت إلى الفصل الأول من هذا الكتاب ستتذكر أن كل الأفعال تصدر عن () main الآن سيبدأ البرنامج فى التنفيذ . .

مثال (١٩-٦): استخدام Class فى برنامج Main

```
#include <iostream.h>

class shape    {
            // DECLARE DATA MEMBERS
            private:
              int length;
              int height;
              int area;

            public:
              void CalcArea(void);
              void ShowArea(void);
            // CREATE A CONSTRUCTOR .
              shape(int l = 0, int h = 0);
            };

// NOW DEFINE THE MEMBER FUNCTIONS
void shape::CalcArea(void)
{
   area = length * height;
}

void shape::ShowArea(void)
{
   cout << "THE AREA IS : " << area << endl;
}

shape::shape(int l, int h)
{
   length = l;
   height = h;
}

main()
{
   int x;
   int y;
```

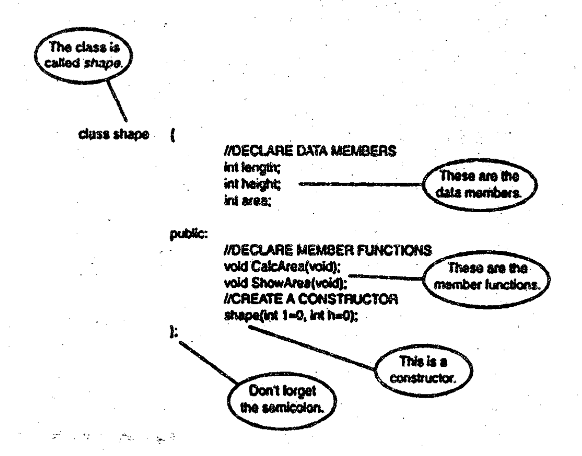
```
    cout << "ENTER THE HEIGHT : ";
    cin >> x;
    cout << "ENTER THE LENGTH : ";
    cin >> y;
    // CREATE AN INSTANCE OF SHAPE
    shape square(y,x);
    //CALL THE MEMBER FUNCTIONS
    square.CalcArea();
    square.ShowArea();
    return(0);
}
```

لقد قمت بتعريف متغيرين فى ( ) main هما y و x . وهذا يستخدم فى تجميع البيانات العامة من العالم الخارجى . أما وقد اطلعت على هذه القيم فسوف تقوم بصنع instance من shape class الذى يسمى مربع (square) ثم تقوم بإرجاع القيم المحملة على المتغيرين y و x إلى data members المطابقة لها داخل class . هذه القيم الجديدة تلغى عمل الإعدادات الإفتراضية .

وبالرجوع ثانياً إلى مثال جهاز التليفزيون فى بداية هذا الفصل ، يكون class هو الدائرة الكهربية لجهاز التليفزيون ، وconstructor هو عملية تركيب الجهاز ، أما instance فهو الجهاز الفعلى النهائى الذى تستطيع لمسه . والقيم التى تسري في constructor هى متطلبات العميل مثل الكابينة الخشبية أو أن تكون شاشة الجهاز ٥٦سم . وكما أوضح نموذج جهاز التليفزيون ، فإنك تستطيع الحصول على أى عدد تريده من instances أو النماذج .

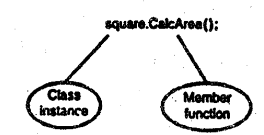
فى اصطلاح الكمبيوتر وبعيداً عن المثال السابق لجهاز التليفزيون ، فإنك تقوم بالآتى : إن class يحتاج إلى الذاكرة بهيئتها بينما يتطلب constructor وضع الكود المناسب داخل هذه الذاكرة ، والكود يأتى من member functions . وفى كل مرة تقوم فيها بهذه العمليات السابقة تصنع unique instance أو نموذجاً فريداً من object ليأخذ مكانه فى ذاكرة الكمبيوتر ، وكل object له صفاته المميزة تبعاً للقيم المدخلة فى constructor والكود الخاص بهذه القيم والذى يأتى بدوره من تعريفات member functions ليدير data members .

لاحظ التركيب المطلوب والموضح فى الشكل الاتى :



شكل يوضح التركيب اللازم لصنع instance من shape

أما وقد قمت بصنع instance من shape (يسمى مربع)، فسوف تقوم الآن بتطبيق member functions التى تدير البيانات الموجودة داخل instance. انظر إلى التركيب الموجود فى الشكل الآتى :



شكل يوضح محدد instance وهو عبارة عن نقطة
وكذلك member functions وقائمة المدخل (وهى void فى هذا المثال)

وهكذا انتهت كل الخطوات اللازمة لكتابة برنامج OOP وأخيراً ستبدأ تنفيذه. هذا هو ما ظهر على الشاشة بعد تنفيذ مثال (١٩-٦).

```
ENTER THE HEIGHT : 6
ENTER THE LENGTH : 7
THE AREA IS : 42
```

تذكر أن أسلوب OOP مثله مثل كل لغات البرمجة يحتاج إلى كود .

الآن وقد تعرفت على كل الخطوات اللازمة لتنفيذ برنامج OOP يجب ألا تنسى أنك لازلت فى حاجة إلى المزيد من التدريب والممارسة ، لذا سنحاول معًا تنفيذ هذا التدريب العملى .

التدريب العملى The Queue (Q):

فى هذا المثال الدراسى فإنك سوف تنظر إلى Q وتحدد الحل المناسب لبرنامج OOP تبعًا للمواصفات المعطاة ، هذا التدريب شائع فى عالم الكمبيوتر لذلك يستحق استيعاب المفهوم والحل . قم بإدخال الكود المعطى فى كل خطوة ثم قم بتصنيفه فهذا سيقلل من عدد الخطأ .

مواصفات Q:

- يجب ألا يحتوى التركيب الجديد على أية عناصر .
- يجب أن يضم عدداً من الأحرف يصل إلى تسعة أحرف .
- يمكن إدخال أى حرف جديد حتى نهاية التركيب .
- لا يجوز الوصول إلا إلى البيانات الموجودة فيبداية التركيب .
- يمكن استعراض Q بالكامل . ستظهر رسالة توضح ما إذا كان خالياً أم لا .
- ستظهر رسالة تنبيه إذا كانت هناك محاولة لإضافة أكثر من تسعة أحرف إلى التركيب .

وبناء على هذه المعلومات ، يمكنك البدء فى كتابة كود برنامج OOP لحل المشكلة .
قم بالرجوع إلى الجزء الأول من هذا الفصل فى كل خطوة وقارنها بمثال shape .



ملحوظة

خطوات إرشادية عن كيفية عمل برنامج OOP:

١- قم بوصف class.

٢- أضف member functions إلى class.

٣- أضف member function prototypes إلى class.

٤- أضف constructor إلى class

٥- عرف الكود المرتبط بكل من member function و constructor.

٦- أضف كود constructor إلى وظيفة ( ) main التى تصنع instance من class. اضف الكود الذى يستخدم member functions ليدير data members داخل class.

الخطوة الأولى:

أولاً تقوم بتعريف class جديد، لا تنسى أن تقوم بجمع وتصنيف هذا البرنامج لتتأكد من خلوه من الأخطاء. لا تقم بتشغيله الآن لأن وظيفة main خالية من الأوامر، لذلك لن يحدث شئ عند التشغيل.

مثال (٧-١٩): الخطوة الأولى من برنامج Q

```
#include <iostream.h>

class Q    {
        // an empty class.
        };
main()
{
   return(0);
}
```

الخطوة الثانية:

ستضيف الآن data members، ومن أسس تنظيم البيانات تستطيع أن تستنتج أن array المكون من تسعة أحرف سيناسب أسس تنظيم البيانات في برنامج Q، ولكنك تحتاج إلى تعيين نهاية Q. لقد قمت بعمل تصميم من data members يسمى back وهو بدوره سيخبرك عن عدد عناصر array الخاضعة للاستخدام حالياً. فإذا أشارت back إلى الصفر فهذا معناه أن Q فارغ، وتأثير ذلك أن array لابد أن يحوى عشرة أحرف وليس تسعة لأنك لا تستخدم العنصر الأول ولكن لابد أن تحتفظ بتخزين تسعة أحرف.



متى أضع متغيرات إضافية؟

من الملاحظ أن الطلبة الجدد يميلون إلى استخدام عدد زائد من المتغيرات، ولكن مع التدريب والخبرة سيأخذ عدد المتغيرات في التناقص.

مثال (١٩-٨): الخطوة الثانية من برنامج Q

```
#include <iostream.h>

class Q    {
        private:
           int back;
           char data[10];
        };

main()
{
      return(0);
}
```

الخطوة الثالثة:

الخطوة الثالثة هي أن تضيف member function prototypes إلى class. وستكون بحاجة إلى member functions لربط البيانات مع Q;I وهذا يسمى joinQ. ولأننا نضيف بيانات، فقد قمت بإعطاء member function مدخل من نوع

character . إنك تحتاج إلى member function ليسمح للبيانات بترك Q وهذا ما أسميته LeaveQ . إن ترك Q يحتاج فقط إلى أمر ولا يحتاج إلى أية معلومات إضافية ولهذا فهو void . وكذلك يلزم وجود member function ليوضح محتويات Q وهذا ما أطلقت عليه ShowQ وهو أيضًا لا يحتاج أية معلومات إضافية ولذا فهو void .

مثال (١٩-٩): الخطوة الثالثة من برنامج Q

```
#include <iostream.h>

class Q {
    private:
      int back;
      char data[10];

    public:
      void JoinQ(char ch);
      void LeaveQ(void);
      void ShowQ(void);
    };

main()
{
   return(0);
}
```

الخطوة الرابعة:

والآن يأتي دور constructor، تقول أسس تنظيم البيانات أن Q الجديد لن يحتوى على أية عناصر . لهذا فنحن مضطرين لقبول ذلك . لقد أطلقت على constructor اسم Q . لماذا؟ لأن constructor يأخذ دائمًا نفس اسم class لهذا فليس هناك مجالاً للاختيار ولا يحتاج constructor إلى أية معلومات إضافية . ولأن Q دائمًا تبدأ خالية لهذا فهى void .

مثال (١٩-١٠): الخطوة الرابعة من برنامج Q

```
#include <iostream.h>

class Q {
    private:
```

```
int back;
    char data[10];

  public:
    void JoinQ(char ch);
    void LeaveQ(void);
    void ShowQ(void);
    Q(void);
  };

main()
{
  return(0);
}
```

الخطوة الخامسة:

والآن ستضيف الكود. س كل ما ستفعله هو أن تقوم بتطبيق كود عادى داخل إطار برنامج OOP. سوف أوضح لك تعريف كل member function بالترتيب.

إن وظيفة constructor هى صنع برنامج Q خالى، ولقد قمت بعمل تصميم ليشير إلى أن back to zero تعنى أن Q خالياً. وهذا هو الكود المطلوب. هذا هو constructor الذى صنع Q جديد. افترض أن back to zero وQ خاليان من العناصر.

```
Q::Q()
{
  back = 0;
}
```

استخدم هذا الكود لإضافة معلومات إلى Q وتمر البيانات إلى متغير من نوع character اسمه ch، وهناك اختبار يطبق لمعرفة ما إذا كانت هناك مسافة فى نهاية Q. تذكر أنه ليس بإمكانك وضع أكثر من عشرة أحرف فى Q، فإذا كانت هناك مساحة فستقوم بزيادة back وتزيد محتويات ch على array. فإذا كان array ممتلئ، فستقوم بعرض رسالة تنبيه توضح أن Q ممتلئ. يسمح member function بزيادة حرف إلى نهايةQ.

```
void Q::JoinQ(char ch)
{
```

```
  if (back < 9)
    {
      back++;
      data[back] = ch;
    }
  else
    cout << "Q is full" << endl;
}
```

إذا كان هناك عنصر سيتم حذفه من Q، فإن أسس تنظيم البيانات ترشدنا إلى أنه يمكن أن يخرج من الأمام فقط، فإنك تستطيع استخدام هذا الكود أيضًا. يجب أولاً أن تعرف ما إذا كان هناك أيه بيانات فى Q، لأنك لا تستطيع أن تترك Q فارغاً. فإذا كان Q حالياً تعلن التحذير ولا تقوم بأى فعل آخر. لكن إذا كان هناك بيانات فى Q استمر فى العمل. فالخدعة تكمن فى أنك تدخل كل البيانات بشكل غير منظم فى مكان واحد مباشرة فى Q، وبهذا تطغى على ما كان مكتوباً بالفعل. وهكذا سيطفو العنصر الذى سيترك Q على السطح. هذا يحدث بواسطة جملة for loop البسيطة، وبعد ذلك يتم خفض قيمة back للدلالة على أن هناك عنصراً تم حذفه من Q.

يسمح member function الآتى لعنصر البيانات الموجود فى مقدمة Q بالخروج. كل العناصر المتبقية فى Q تحركت بمقدار مكان واحد.

```
void Q::LeaveQ(void)
{
  int index;

  if (back > 0)
  {
    for (index = 1; index < back; index++)
    {
      data[index] = data[index + 1];
    }
    back--;
  }
  else
  {
    cout << "Q is already empty" << endl << endl;
  }
}
```

ويقوم آخر member function بعرض البيانات على الشا' ة فلو أن back هو الصفر فهذا يعنى أن Q فارغ، ولهذا ظهرت رسالة مناسبة . وبالتبادل، يستخدم for loop لعرض عناصر array ومحتوياتها على الشاشة . ويلاحظ أن loop تبدأ بالعنصر الثانى وعنوانه 1 حيث أن العنصر الأول [0] يستخدم كعلامة على خلو Q من العناصر وتستخدم member function لعرض محتويات Q .

```
void Q::ShowQ(void)
{
   int x;

   if (back == 0)
   {
      cout << "Q is empty" << endl << endl;
   }
   else
   {
      for (x = 1; x <= back; x++)
      {
         cout << data[x] << TAB;
      }
      cout << endl << endl;
   }
}
```

هذا هو البرنامج ولكن تذكر أنه لن يعطى أية نتائج إذا تم تشغيله .

مثال (١٩-١١): الخطوة الخامسة من برنامج Q

```
#include <iostream.h>

#define TAB '\t'

class Q {
        private:
           int back;
           char data[10];

        public:
           void JoinQ(char ch);
```

```
        void LeaveQ(void);
             void ShowQ(void);
             Q(void);
        };

// This is the constructor used
// to set up an empty Q.
// Assume that if back is zero, Q is empty
Q::Q()
{
   back = 0;
}

// This member function is used
// to show the contents of the Q
void Q::ShowQ(void)
{
   int x;

   if (back == 0)
   {
      cout << "Q is empty" << endl << endl;
   }
   else
   {
      for (x = 1; x <= back; x++)
      {
         cout << data[x] << TAB;
      }
      cout << endl << endl;
   }
}

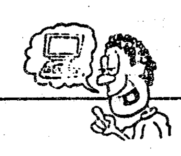
// This member function allows
// a letter to be added to the
// end of the Q
void Q::JoinQ(char ch)
{
```

```
if (back < 9)
   {
      back++;
      data[back] = ch;
   }
   else
      cout << "Q is full" << endl;
}

// This member function allows
// the data item at the front of
// the Q to leave. All remaining
// items in the Q are moved up one place
void Q::LeaveQ(void)
{
   int index;

   if (back > 0)
   {
      for (index = 1; index < back; index++)
      {
         data[index] = data[index + 1];
      }
      back--;
   }
   else
   {
      cout << "Q is already empty" << endl << endl;
   }
}

main()
{
   return(0);
}
```



استخدام void Constructors

إن استخدام void constructors شاق إلى حد ما. لاحظ التركيب الصحيح.

Q one;

كن حذراً فليس هناك أقواساً بعد one.

الخطوة السادسة:

لإثبات أن البرنامج يعمل بالفعل قمت بعمل قائمة من ثلاثة أحرف باستخدام joinQ وهو member function، يعرض النتائج بعد كل استدعاء بواسطة ShowQ. ثم أخرجت البيانات. Q باستخدام LeaveQ. وقبل التنفيذ قمت بعمل instance من Q أسميته one بإستخدام constructor.

```
main()
{
   Q One;
   One.ShowQ();
   One.JoinQ('Z');
   One.ShowQ();
   One.JoinQ('A');
   One.ShowQ();
   One.JoinQ('K');
   One.ShowQ();
   One.LeaveQ();
   One.ShowQ();
   One.LeaveQ();
   One.ShowQ();
   One.LeaveQ();
   One.ShowQ();
   return(0);
}
```

تأكد من إكتمال برنامج Q الآن . لقد أوضحنا تفصيلاً هذا التطبيق العملي، والآن دعنا نقوم بالتنفيذ في مثال (١٩-١٢). يجب أن يظهر الآتي على الشاشة.

```
Q is empty
Z
Z  A
Z  A  K
A  K
K
Q is empty
```

مثال (۱۹-۱۲): برنامج Q

```
#include <iostream.h>

#define TAB '\t'

class Q {
        private:
           int back;
           char data[10];

        public:
           void JoinQ(char ch);
           void LeaveQ(void);
           void ShowQ(void);
           Q(void);
     };

// This is the constructor used
// to set up an empty Q.
// Assume that if back is zero, Q is empty
Q::Q()
{
   back = 0;
}

// This member function is used
// to show the contents of the Q
void Q::ShowQ(void)
{
   int x;

   if (back == 0)
   {
      cout << "Q is empty" << endl << endl;
   }
   else
   {
      for (x = 1; x <= back; x++)
```

```
{
        cout << data[x] << TAB;
      }
      cout << endl << endl;
   }
}

// This member function allows
// a letter to be added to the
// end of the Q
void Q::JoinQ(char ch)
{
   if (back < 9)
   {
      back++;
      data[back] = ch;
   }
   else
      cout << "Q is full" << endl;
}

// This member function allows
// the data item at the front of
// the Q to leave. All remaining
// items in the Q are moved up one place
void Q::LeaveQ(void)
{
   int index;

   if (back > 0)
   {
      for (index = 1; index < back; index++)
      {
         data[index] = data[index + 1];
      }
      back--;
   }
   else
```

```
    {
        cout << "Q is already empty" << endl << endl;
    }
}

main()
{
   Q One;
   One.ShowQ();
   One.JoinQ('Z');
   One.ShowQ();
   One.JoinQ('A');
   One.ShowQ();
   One.JoinQ('K');
   One.ShowQ();
   One.LeaveQ();
   One.ShowQ();
   One.LeaveQ();
   One.ShowQ();
   One.LeaveQ();
   One.ShowQ();
   return(0);
```

برنامج Q بعد التعديل:

لقد أوضح مثال (١٩-١٢) أن برنامج Q الذى يستخدم أسلوب OOP يعمل بكفاءة، ولكنه إلى حد ما ثابت. سنقوم ببعض التعديلات التى تجعله متفاعل. ولعمل ذلك فإنك تحتاج إلى لمس OOP bit لأنك تعلم أنها تعمل. كل ما تحتاجه هو أن تضيف menu و switch case كما فى الفصل الثامن. *استخدمات جملة switchcase* وتستطيع رؤية Q بعد ذلك أو الإضافة إليه أو تركه أثناء وقت التشغيل. انظر مثال (١٩-١٣).

مثال (١٩-١٢): استعراض أسلوب OOP: برنامج Q بعد التعديل

```
#include <iostream.h>
#include <stdlib.h>
```

```
#define TAB '\t'

class Q {
        private:
           int back;
           char data[10];

        public:
           void JoinQ(char ch);
           void LeaveQ(void);
           void ShowQ(void);
           Q(void);
      };

// This is the constructor used
// to set up an empty Q.
// Assume that if back is zero, Q is empty
Q::Q()
{
   back = 0;
}

// This member function is used
// to show the contents of the Q
void Q::ShowQ(void)
{
   int x;

   if (back == 0)
   {
      cout << "Q is empty" << endl << endl;
   }
   else
   {
      for (x = 1; x <= back; x++)
      {
         cout << data[x] << TAB;
      }
```

```
  cout << endl << endl;
   }
}

// This member function allows
// a letter to be added to the
// end of the Q
void Q::JoinQ(char ch)
{
   if (back < 9)
   {
      back++;
      data[back] = ch;
   }
   else
      cout << "Q is full" << endl;
}

// This member function allows
// the data item at the front of
// the Q to leave. All remaining
// items in the Q are moved up one place
void Q::LeaveQ(void)
{
   int index;

   if (back > 0)
   {
      for (index = 1; index < back; index++)
      {
         data[index] = data[index + 1];
      }
      back--;
   }
   else
   {
      cout << "Q is already empty" << endl << endl;
   }
```

```
}

char menu(void);

main()
{
  char ch;
  char flipper;
  Q One;

  while (1)
  {
    ch = menu();
    switch(ch)
    {
      case '1' :  cout << "Enter the character : ";
                  cin >> flipper;
                  One.JoinQ(flipper);
                  break;
      case '2' :  One.LeaveQ();
                  break;
      case '3' :  One.ShowQ();
                  break;
      case '4' :  exit(0);
    }
  }
  return(0);
}

char menu(void)
{
  char choice;

  cout << "1...Join the Q" << endl;
  cout << "2...Leave the Q" << endl;
  cout << "3...Show the Q" << endl;
  cout << "4...Quit the program" << endl << endl;
  cout << "Enter your choice : ";
```

```
cin >> choice;
   return (choice);
}
```

إنك لا تحتاج إلى Constructor

من الناحية الفنية فانت لا تحتاج إلى تحديد constructor لأنك إذا قمت بحذفه فسيقوم compiler بعمل أخر افتراضياً غير مرئى لك. فى هذه الحالة ستكون قد قمت بعمل نموذج من class مثله مثل برنامج Q. هذا الأسلوب يناسب البرامج البسيطة إلا أنك لن تستطيع التحكم فى بداية object.

أهم الدروس المستفادة:

- إن أسلوب object-oriented programming يسمى أيضًا OO أو OOP .
- يعرف class بأنه تركيب البيانات الذى يحتوى على كل ما يلزم لتخزين وإدارة البيانات
- المتغير المعرف داخل class يسمى data member .
- الوظائف التى تعالج البيانات تسمى member functions .
- يرتبط كل من data members و member functions إرتباطًا وثيقًا داخل كيان واحد (a class)، ولهذا ليس لأحدهما تأثير بدون الآخر. وهذا هو ما يسمى encapsulation .
- أى public member فى class يمكنه الدخول من خارج class .
- أى private member فى class لا يمكنه الدخول من خارج class، ويمكنه فقط الدخول بواسطة public members .
- هناك اصطلاح فى لغة C++ على إضافة خاصية private إلى كل data members وما دون ذلك يعد من أساليب فى البرمجة السيئة .

- يعتبر كل members داخل class إفتراضياً private .
- يعد constructor نوعاً خاصاً من member functions يستخدم في صن instance من class . هذا يجعل الذاكرة تخصص مكان لاستخدام class instance . تسمى الذاكرة المادية object .


الفصل الحادي عشر
المدمر

الفصل الحادى عشر
المدمر

وفى هذا الفصل ستتعرف أكثر على هذه المفاهيم بالإضافة إلى المفهوم الجديد؛ Destructor .

أهمية Destructor:

إن أهمية destructor تكمن فى قدرته على إنهاء وتدمير أى object بطريقة منظمة . فالبرامج التى كتبتها فى الفصل التاسع عشر تبدو خالية من destructor، ولكن هذا خطأ، لأنك حين تقوم بعمل object بواسطة constructor، يتم عمل destructor إفتراضى بشكل تلقائى . ولكنك تستطيع إحلال destructor من صنعك محل destructor الإفتراضى وسنضرب مثلاً على user-defined destructor الذى يقوم مستخدم الجهاز بإنشائه .

ما هو Destructor؟

إن المثال العملى يدور حول برنامج يتناول كيفية التعامل مع string ويمثل أساساً للتعامل مع الأربعة أمثلة العملية الآتى ذكرها فى هذا الفصل .

ولا يختلف تعريف destructor كثيراً عن تعريف constructor إلا فى أنه يسبق بالرمز ~ . ويتواجد prototype جنباً إلى جنب مع public member functions الآخرين داخل class .

```
~StringThings(void);
```

وهكذا فإن التعريف يتشابه مع مثيله الذى يخص constructor من حيث وضع اسم class، و scope resolution operator، وأخيراً اسم destructor . ودائماً اسم destructor هو نفس اسم class . لقد قمت بوضع رسالة معينة داخل destructor لتوضيح كيفية عمله، ويمكنك إحلالها بأى كود مقارب، مثل إغلاق أية ملفات مفتوحة .

```
StringThings::~StringThings(void)
{
cout << endl << "THAT'S ALL FOLKS" << endl << endl;
}
```

والسؤال الآن، هل يمكن أن يخلو برنامج من destructor؟ والإجابة هى نعم، لكن لفترة قصيرة . تذكر أن constructor يمنع الذاكرة من صنع object، فكلما تم عمل object نقصت مساحة الذاكرة المتاحة . ومهمة destructor هى تحرير الذاكرة لإعادة

استخدامها. والفشل فى إتمام هذه المهمة يؤدى إلى ما يعرف باسم memory leak (امتلاء الذاكرة وتسرب طاقتها)، وكلما إزدادت مرات الفشل كلما تأثرت الذاكرة وضعفت وأصبح الجهاز فى حاجة إلى إعادة تشغيل. ولكن طالما أن برنامجك يعمل جيداً، سيقوم destructor الإفتراضى بوظيفة تحرير الذاكرة.

إن المثال العملى- الذى سيرد ذكره- لديه member function يقوم بايجاد أول حرف من string ثابت. فهذه العملية هى مجرد نسخ أول حرف من string (وهو عبارة عن array of characters) داخل data member وهو متغير من نوع char. محاولة كشف الخطأ مطلوبة لأنه يمكن أن يوجد string فارغاً، ولا يمكن أن يوجد character فارغاً. ولقد قمت باستخدام "?" بدلاً من الحرف الأول الغير موجود. وكل هذا سيتضح من خلال member function الآتى:

```
void StringThings::First(void)
{
if (strlen(str) < 1)
FirstLetter = '?';
      else
            FirstLetter = str[0];
}
```

وتتمثل وظيفة show ( ) هنا فى عرض محتويات data member المسمى FirstLetter على الشاشة.

```
void StringThings::Show(void)
{
   cout << FirstLetter << endl;
}
```

يقوم constructor بإنشاء صيغ stringThing object وكذلك إدخال string إلى private. لاحظ استخدام strcpy المأخوذ من string.h library.

```
StringThings::StringThings(char s[20])
{
   strcpy(str,s);
}
```

تنفيذ البرنامج:

هذا هو ما سيظهر على الشاشة عند تنفيذ البرنامج.

```
P

THAT'S ALL FOLKS
```

لقد ظهر الحرف P على الشاشة بواسطة ( ) show. ولكن السطر الثانى يحتاج إلى مزيد من الإيضاح. فكل مرة ينتهى فيها عمل object، يأتى دور destructor فى تحرير الذاكرة مرة أخرى مما عليها. كما يتضمن destructor هذا السطر أيضاً.

```
cout << endl << "THAT'S ALL FOLKS" << endl << endl;
```

ومن هنا تظهر الرسالة على الشاشة.

مثال (٢٠-١) يشرح البرنامج كاملاً.

مثال (٢٠-١): كيفية إنشاء Destructor:

```
#include <iostream.h>
#include <string.h>

class StringThings {
                private:
                        char str[20];
                        char FirstLetter;

                public :
                        void First(void);
                        void Show(void);
                        StringThings(char s[20]);
                        ~StringThings(void);
        };

void StringThings::First(void)
{
    if (strlen(str) < 1)
        FirstLetter = '?';
    else
        FirstLetter = str[0];
}
```

```
void StringThings::Show(void)
{
     cout << FirstLetter << endl;
}

StringThings::StringThings(char s[20])
{
     strcpy(str,s);
}

StringThings::~StringThings(void)
{
     cout << endl << "THAT'S ALL FOLKS" << endl << endl;
}

main()
{
     StringThings Test("PAUL");
     Test.First();
     Test.Show();
     return(0);
}
```

اختبر معلوماتك عن Destructor:

والآن يأتي دور التدريبات العملية. سنبدأ بالأيسر ثم نترقى تدريجياً إلى الأكثر صعوبة.

التدريب العملي الأول:

المشكلة:

نجد في المثال (٢٠-١) أن عملية إدخال string قد تمت عن طريق البرنامج نفسه أثناء compile time. حاول تعديل البرنامج ليسمح للمستخدم أن يقوم بإدخال string أثناء وقت التشغيل (runtime). قم بتعديل destructor ليعرض على الشاشة المدخلات بالكامل أثناء عملية تدمير object.

الحل:

لم يكن من الضرورى استخدام أسلوب OOP فى التعديل الأول. فمن خلال برنامج main يستطيع المستخدم إدخال string، ثم يمكن قراءته بواسطة cin.getline.

```
cout << "Enter the string : ";
cin.getline(data,19);
```

أما لتعديل destructor. فإنك فقط ستعيد كتابة أمر caut مرة أخرى ..

```
StringThings::~StringThings(void)
{
    cout << endl << "THE FULL STRING WAS " << str;
    cout << endl << "THAT'S ALL FOLKS" << endl << endl;
}
```

حاول تنفيذ البرنامج فى مثال (٢٠-٢).

مثال (٢٠-٢): كيفية تنظيم Destructor:

```
#include <iostream.h>
#include <string.h>

class StringThings {
                private:
                    char str[20];
                    char FirstLetter;

                public :
                    void First(void);
                    void Show(void);
                    StringThings(char s[20]);
                    ~StringThings(void);
            };

void StringThings::First(void)
{
    if (strlen(str) < 1)
        FirstLetter = '?';
    else
        FirstLetter = str[0];
```

```
}

void StringThings::Show(void)
{
    cout << FirstLetter << endl;
}

StringThings::StringThings(char s[20])
{
    strcpy(str,s);
}

StringThings::~StringThings(void)
{
    cout << endl << "THE FULL STRING WAS " << str;
    cout << endl << "THAT'S ALL FOLKS" << endl << endl;
}

main()
{
    char data[20];

    cout << "Enter the string : ";
    cin.getline(data,19);
    StringThings Test(data);
    Test.First();
    Test.Show();
    return(0);
}
```

المثال العملى الثانى:

المشكلة:

حاول تعديل مثال (٢٠-١) عن طريق كتابة member function جديد لإيجاد الحرف الأخير من string . فإذا كان string طوله حرفاً واحداً . فسيعتبر هذا هو الحرف الأول والأخير فى نفس الوقت . أطلق على ( ) show اسم ShowFirst ومهمتها عرض

الحرف الأول على الشاشة . اكتب member function جديد يكون اسمه ShowLast، ليعرض الحرف الأخير من string على الشاشة .

الحل:

لقد أضفت data member تابعة للقطاع private ومن نوع character، وأطلقت عليها اسم Lastletter سوف تحمل data member الحرف الأخير من المدخل string أما member function المسمى last سيطابق First . في الحقيقة هناك سطر واحد مختلف، وهذا هو السطر الذي يجد آخر حرف . تستخدم جملة strlen (str)-1 تستخدم لإيجاد عنوان الحرف الأخير من string . وهذه القيمة متضمنة في array من نوع char اسمها str. وهكذا يمكن نسخ آخر حرف في LastLetter .

```
void StringThings::Last(void)
{
    if (strlen(str) < 1)
        LastLetter = '?';
    else
        LastLetter = str[strlen(str)-1];
}
```

إن ShowLast member function يماثل ShowFirst باستثناء إمكانية عرض data member LastLetter .

ماهو strlen؟

توجد strlen ( ) في string.h library، ومهمتها هي إظهار طول string في شكل مخرجات على الشاشة. القيمة الناتجة عبارة عن integer غير محدد وذلك لأنه ليس هناك string طوله بالإشارة السالبة. تذكر أن string عبارة عن array of character وأنك تتعامل مع العنوان عندما تريد البحث عن أرقام مفردة داخل string . ويقل العنوان دائمًا بمقدار واحد عن عدد العناصر. ولهذا يجب أن تستخدم تعبير strlen (str)-1 لكي تحدد العنوان الخاص بآخر حرف داخل string.

مثال (٢٠-٢٠): String Things في برنامج Object-Oriented:

```
#include <iostream.h>
#include <string.h>

class StringThings {
            private:
```

```
        char str[20];
        char FirstLetter;
        char LastLetter;

    public :
        void First(void);
        void Last(void);
        void ShowFirst(void);
        void ShowLast(void);
        StringThings(char s[20]);
        ~StringThings(void);
    };

void StringThings::First(void)
{
    if (strlen(str) < 1)
        FirstLetter = '?';
    else
        FirstLetter = str[0];
}

void StringThings::Last(void)
{
    if (strlen(str) < 1)
        LastLetter = '?';
    else
        LastLetter = str[strlen(str)-1];
}

void StringThings::ShowFirst(void)
{
    cout << FirstLetter << endl;
}

void StringThings::ShowLast(void)
{
    cout << LastLetter << endl;
}
```

```
StringThings::StringThings(char s[20])
{
    strcpy(str,s);
}

StringThings::~StringThings(void)
{
    cout << endl << "THE FULL STRING WAS " << str;
    cout << endl << "THAT'S ALL FOLKS" << endl << endl;
}

main()
{
    char data[20];

    cout << "Enter the string : ";
    cin.getline(data,19);
    StringThings Test(data);
    Test.Last();
    Test.ShowLast();
    return(0);
}
```

التدريب العملى الثالث:

المشكلة:

أكتب برنامج أيقوم بعكس ترتيب الحروف داخل string ، وإذا كان طول string حرفاً واحداً، فإن هذا الحرف هو الأول والأخير فى نفس الوقت. اعرض string الأصلى وبعد عكس ترتيبه على الشاشة.

الحل:

نحتاج إلى member function اسمه Reverse . وهذا الإسم يتناسب مع الوظيفة التى يؤديها. أولاً ستحدد طول string . وسيكون عدد الحروف مساو لعدد العناصر. ولكن لأنك تتعامل مع array ، فستنقص واحد بسبب العنوان. وإذا كان string فارغاً، فستقوم فقط بنسخ str داخل Reverse .

ستجد الطول وستقوم بإنقاصه واحداً، وهكذا سينتج الرقم 2 وهو قيمة last .

```
RevStr[index] = str[last - index];
```

وفى أثناء loop سيكون رقم الفهرس القيم 2 1 0 فى حين أن رقم الفهرس last يأخذ قيم 0 1 2 . وأخيراً يجب أن تضع NULL terminator فى نهاية string المنعكس . ولكنك تحتاج إلى نسخ string فارغ لأن كل data members عبارة عن strings وكل strings لديها NULL terminator فى آخرها . فأنت فى الحقيقة تقوم بطبع NULL terminator .

```
void StringThings::Reverse(void)
{
    unsigned int index;
    unsigned int last;

    last = strlen(str);
    last--;
    if (strlen(str) < 1)
        strcpy(RevStr,str);
    else
    {
        for (index = 0; index < strlen(str); index++)
        {
            RevStr[index] = str[last - index];
        }
        RevStr[++last] = NULL;
    }
}
```

وهذا هو عرض التدريب العملى الثالث بشكل تفصيلى .

مثال (٤-٢٠): مزيد من التدريب على أسلوب OPP:

```
#include <iostream.h>
#include <string.h>

class StringThings {
            private:
                char str[20];
                char RevStr[20];
```

```
                    public :
                    void Reverse(void);
                    void ShowRevStr(void);
                    StringThings(char s[20]);
                    ~StringThings(void);
                };

void StringThings::Reverse(void)
{
    unsigned int index;
    unsigned int last;

    last = strlen(str);
    last--;
    if (strlen(str) < 1)
        strcpy(RevStr,str);
    else
    {
        for (index = 0; index < strlen(str); index++)
        {
            RevStr[index] = str[last - index];
        }
        RevStr[++last] = NULL;
    }
}

void StringThings::ShowRevStr(void)
{
    cout << RevStr << endl;
}

StringThings::StringThings(char s[20])
{
    strcpy(str,s);
}

StringThings::~StringThings(void)
```

```
{
    cout << endl << "THE FULL STRING WAS " << str;
    cout << endl << "THAT'S ALL FOLKS" << endl << endl;
}

main()
{
    char data[20];

    cout << "Enter the string : ";
    cin.getline(data,19);
    StringThings Test(data);
    Test.Reverse();
    Test.ShowRevStr();
    return(0);
}
```

التدريب العملى الرابع:

المشكلة:

اكتب برنامجاً بأسلوب OPP لحساب عدد الحروف المتحركة داخل string .

الحل:

إن هذا التدريب العملى سيجعلنا نتعرف على جمل if else و case. وهذا هو member function الذى يتضمن الكود المطلوب . فهناك جملة for loop التى يتم من خلالها المرور على كل حرف فى data member str . ويقوم switch case بالتأكد من أن هذا الحرف متحركاً ، وحينما يجد حرفاً متحركاً يزداد عداد data member بمقدار عدد واحد.لأن العداد يحمل الرقم صفر طالما أنه لا يزال داخل constructor .

```
void StringThings::VowelCount(void)
{
    unsigned int index;

    for (index = 0; index < strlen(str); index++)
    {
        switch (str[index])
        {
```

```
            case 'A' :
            case 'a' :  count++;
                        break;
            case 'E' :
            case 'e' :  count++;
                        break;
            case 'I' :
            case 'i' :  count++;
                        break;
            case 'O' :
            case 'o' :  count++;
                        break;
            case 'U' :
            case 'u' :  count++;
                        break;
        }
    }
}
```

تذكر الفصل الثامن " استخدامات جملة switch case " حينما كنا نقوم بتحويل كل شئ إلى حروف استهلالية كبيرة . فهذه طريقة أخرى لها . وعليك الإختيار .

حاول تنفيذ البرنامج فى المثال (٢٠-٥) الآتى وقم بحساب عدد الحروف المتحركة الموجودة فى اسمك .

مثال (٢٠-٥): كيفية حساب عدد الحروف المتحركة:

```
#include <iostream.h>
#include <string.h>

class StringThings {
            private:
                  char str[20];
                  int count;

            public :
                  void VowelCount(void);
                  void ShowCount(void);
                  StringThings(char s[20]);
```

```
                ~StringThings(void);
            };

void StringThings::VowelCount(void)
{
    unsigned int index;

    for (index = 0; index < strlen(str); index++)
    {
        switch (str[index])
        {
        case 'A' :
        case 'a' :   count++;
                     break;
        case 'E' :
        case 'e' :   count++;
                     break;
        case 'I' :
        case 'i' :   count++;
                     break;
        case 'O' :
        case 'o' :   count++;
                     break;
        case 'U' :
        case 'u' :   count++;
                     break;
        }
    }
}

void StringThings::ShowCount(void)
{
    cout << "The number of vowels is : "
<< count << endl;
}

StringThings::StringThings(char s[20])
{
```

```
    strcpy(str,s);
    count = 0;
}

StringThings::~StringThings(void)
{
    cout << endl << "THE FULL STRING WAS " << str;
    cout << endl << "THAT'S ALL FOLKS" << endl << endl;
}

main()
{
    char data[20];

    cout << "Enter the string : ";
    cin.getline(data,19);
    StringThings Test(data);
    Test.VowelCount();
    Test.ShowCount();
    return(0);
}
```

تعرف على المزيد عن Overloaded Functions:

لقد تناولنا فى الفصل السابع عشر "عرض مفهوم Overloaded Functions" موضوع Overloaded Functions التى تعرف بأنها class constructors ويمكن أن يكون لديك العديد من constructors التى تستطيع إنشاء object بطرق متعددة تعتمد على عناصر الإدخال. وبعبارة أخرى فإن class الواحد يمكن أن يسلك بعدة طرق معتمداً على أى constructor تم استخدامه لإنشاء object.

ما هى Multiple Constructors؟

ولتوضيح ما هى multiple constructors فلابد من دراسة شكل ما. ففى أساليب البرمجة التقليدية القديمة كنا نكتب ثلاث وظائف منفصلة ونتعامل مع الثلاثة أشكال على أنها عناصر مختلفة تماماً. أما فى أسلوب البرمجة الحديث OOP فإنك تتعامل معها على أساس أنها عناصر تنتمى لعائلة واحدة.

بدء تشغيل Constructor:

يضم المثال (٢٠-٦) ثلاثة constructors، يختلف كل منها عن الآخر. الأول به مدخل واحد ينتمى إلى الدائرة. والثانى به مدخلان وينتمى إلى المستطيل، أما الثالث فبه ثلاثة مدخلات وينتمى إلى الصندوق. وكلها تنتمى لعائلة الأشكال.

```
shape(int r);
shape(int l, int h);
shape(int l, int h, int d);
```

يمكن أن يكون لديك أى عدد تريده من constructors فهذا يعتمد على درجة تعقيد class. وحينما تقوم بتعريف constructors يمكنك أن تحدد المعادلة الحسابية الصحيحة لكل منها. ولقد قمت بتحديد ست data members وأوضحت وظيفة كل منها ويظهر هذا فى القائمة التالية:

- الطول length: يحمل البيانات الخاصة بأبعاد الطول ويستخدم فى إيجاد حل أشكال المستطيل والصندوق.
- الإرتفاع height: يحمل البيانات الخاصة بأبعاد الإرتفاع ويستخدم فى إيجاد حل أشكال المستطيل والصندوق.
- العمق depth: يحمل البيانات الخاصة بأبعاد العمق ويستخدم فقط لحل شكل الصندوق.
- القطر radius: يحمل البيانات الخاصة بأبعاد القطر، ويستخدم فقط لإيجاد حل شكل الدائرة.
- الإجابة answer: يحمل النتائج الحسابية للدائرة أو المستطيل أو الصندوق.
- which: هى عبارة عن data member يستخدم كعلامة لتمييز الأوامر الصحيحة للـ object المنشأ. ويمكن أن تأخذ قيمة 0 للدائرة، و1 للمستطيل و2 للصندوق.

إن constructor الخاص بالدائرة يقوم بنقل القيمة الخارجية r إلى data member باسم radius ويجعل العلامة تشير إلى الصفر لتدل على أن object المنشأ سيكون دائرة.

```
shape::shape(int r)
{
    radius = r;
```

```
which =0;
}
```

بينما يقوم constructor الخاص بالمستطيل بنقل القيمة الخارجية l و h إلى data membes المسماه length وheight ويجعل العلامة تشير إلى 1 لتدل على أن object المنشأ سيكون مستطيلاً :

```
shape::shape(int l, int h)
{
    length = l;
    height = h;
    which = 1;
}
```

ويقوم constructor الخاص بالصندوق بنقل القيمة الخارجية l و h و d إلى data members المسماه length، height، depth أو سيجعل العلامة تشير إلى 2 لتدل على أن object المنشأ سيكون صندوقاً :

```
shape::shape(int l, int h, int d)
{
    length = l;
    height = h;
    depth = d;
    which = 2;
}
```

إن member function الوحيد الذى يستطيع القيام بالعمليات الحسابية هو Calc . تذكر أن switch هى العلامة التى تحمل القيمة 2 أو 1 و0 . يستخدم هذا فى switch case لتطبيق معادلة object الصحيحة .

```
void shape::Calc(void)
{
    switch(which)
    {
    case 0 : answer = radius * radius * 3.14;
          break;
    case 1 : answer = length * height;
          break;
    case 2 : answer = length * height * depth;
          break;
```

```
   }
}
```

وتشبه وظيفة ( ) Show هنا وظيفة Calc . ومرة أخرى يقوم جملة witch case بتحريك العلامة لتوضح طبيعة object وإظهار الرسالة المناسبة .

```
void shape::Show(void)
{
    switch(which)
    {
    case 0 : cout << "The circles area is : "
<< answer;
            cout << endl << endl;
            break;
    case 1 : cout << "The rectangles area is : "
<< answer;
            cout << endl << endl;
            break;
    case 2 : cout << "The boxes volume is : "
<< answer;
            cout << endl << endl;
            break;
    }
}
```

عودة ثانية إلى Destructor:

سنستخدم destructor لنعطى للمستخدم رسالة عند غلق object . يسمح بوجود destructor واحد فى كل class . سيقوم compiler بعمل destructor افتراضى إذا لم تقم أنت بإنشائه . وبإمكانك تعديل destructor وسيحل محل الأول بمهارة . وبالتالى لن يكون لديك سوى destructor وحيد .

تنفيذ برنامج الأشكال:

الآن وقد عرفت كيفية عمل البرنامج ، حاول تنفيذه وانتظر النتائج . سأساعدك بعرض المخرجات التى ستظهر على الشاشة ، وعليك استنتاج لماذا ظهرت بهذا الشكل .

```
The circles area is : 4
The rectangles area is : 24
```

The boxes volume is : 12.56

THAT'S ALL FOLKS
THAT'S ALL FOLKS
THAT'S ALL FOLKS

قم بالإجابة عن السؤال الآتى قبل البدء فى تنفيذ مثال (٢٠-٦) .

لماذا ظهر سطر THAT'S ALL FOLKS ثلاث مرات؟

مثال (٢٠-٦)، كيفية عمل Multiple Constructors

```
#include <iostream.h>

class shape{
            public:
                  int length;
                  int height;
                  int depth;
                  int radius;
                  double answer;
                  int which;

            public:
                  void Calc(void);
                  void Show(void);
                  shape(int r);
                  shape(int l, int h);
                  shape(int l, int h, int d);
                  ~shape(void);
            };

shape::shape(int r)
{
     radius = r;
     which =0;
}

shape::shape(int l, int h)
{
```

```
length = l;
    height = h;
    which = 1;
}

shape::shape(int l, int h, int d)
{
    length = l;
    height = h;
    depth = d;
    which = 2;
}

shape::~shape(void)
{
    cout << "THAT'S ALL FOLKS" << endl;
}

void shape::Calc(void)
{
    switch(which)
    {
    case 0 : answer = radius * radius * 3.14;
            break;
    case 1 : answer = length * height;
            break;
    case 2 : answer = length * height * depth;
            break;
    }
}

void shape::Show(void)
{
    switch(which)
    {
    case 0 : cout << "The circles area is : "
<< answer;
            cout << endl << endl;
```

```
        break;
    case 1 : cout << "The rectangles area is : "
<< answer;
            cout << endl << endl;
            break;
    case 2 : cout << "The boxes volume is : "
<< answer;
            cout << endl << endl;
            break;
    }
}

main()
{
    shape square(2,2);
    shape cube(2,3,4);
    shape circle(4);
    square.Calc();
    cube.Calc();
    circle.Calc();
    square.Show();
    cube.Show();
    circle.Show();
    return(0);
}
```

إجابة سؤال لماذا ظهر سطر THAT'S ALL FOLKS ثلاث مرات؟

يترك destructor الفرصة حتى يتم object عمله، ولدينا ثلاثة objects في البرنامج يستدعي destructor الثلاثة objects في نهاية البرنامج حتى يتجنب حدوث تسرب في طاقة الذاكرة. بناء على ذلك ظهرت عبارة THAT'S ALL FOLKS ثلاث مرات.

أهم الدروس المستفادة:

- إن وظيفة destructor هي إنهاء عمل object بطريقة منظمة وهكذا يحرر الظاهرة ويتجنب حدوث ظاهرة memory leaks.

- حينما يتم إنشاء object جديد من خلال constructor، يتم تلقائيًا عمل destructor افتراضي.
- يمكنك إبطال عمل destructor افتراضي وإضافة destructor جديد من إنشائك.
- يتشابه تعريف destructor مع تعريف constructor من حيث أنه public member function إلا أنه يسبق بالرمز ~.
- يتكون كل string من array of characters يوجد في نهايتها NULL terminator.
- يمكن أن يكون لديك أكثر من constructor لإنشاء object بطرق مختلفة على أساس signature الخاص بكل constructor.
- يمكن أن يكون لديك أي عدد من constructors طالما أن لكل منها signature مختلف.
- لا يمكن أن يكون لديك أكثر من destructor واحد نشط في كل class.
- يترك destructor الفرصة حتى يتم object عمله. فإذا كان لدينا ثلاثة objects في البرنامج، سيستدعى destructor كل واحد منها على حده.



الفصل الثانى عشر
الملفات

الفصل الثانى عشر
الملفات

يتناول هذا الفصل النقاط التالية :

- نظرية file handling.
- حفظ واسترجاع text files.
- حفظ واسترجاع binary files.
- حفظ واسترجاع الملفات المكونة من مجموعة من السجلات.

عرض نظرية File Handling:

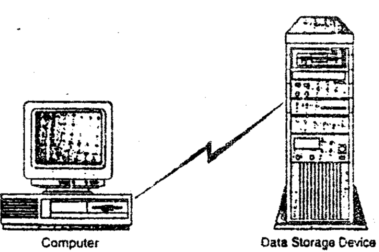
لقد استلزم إدخال وإخراج أية بيانات فى الفصول السابقة استخدام لوحة المفاتيح والشاشة . ومع ذلك يمكنك أيضًا القراءة والكتابة من خلال استخدام الأجهزة الخارجية المتصلة بالكمبيوتر مثل آلة الطباعة والمودم ومحرك الأقراص . ولتوضيح كيفية إرسال واستقبال بيانات من الأجهزة الخارجية ، سأشرح تقنية تسمى file handling، وهى عبارة عن استخدام محرك الأقراص المتصل بجهاز الكمبيوتر لتخزين البيانات . ويعتبر ذلك شكلاً من أشكال التخزين الأكثر استخدامًا فى عالم الكمبيوتر . ولقد لاحظت فى الفصول السابقة أنه يجب عليك تسجيل البيانات فى كل مرة تقوم فيها بتشغيل البرنامج . أما من خلال تقنية file handling يمكنك تخزين البيانات للأبد على محرك الأقراص ، وذلك بعد تسجيلك للبيانات أو حصولك عليها باستخدام الأجهزة الخارجية .

عرض مفهوم Secondary Storage:

إن Secondary storage مو عبارة عن أى جهاز يمكنه تخزين البيانات الإلكترونية التى يمكن للكمبيوتر أن يستخدمها . ومن أمثلة هذه الأجهزة محركات الأقراص الثابتة وأشرطة التسجيل . فالبيانات التى يتم تخزينها على إحدى هذه الأجهزة لا تختفى عند إغلاق جهاز الكمبيوتر . بل تبقى ليتم استخدامها مرة ثانية .

تخزين كميات هائلة من البيانات:

يمكن فى الواقع تخزين كميات ضخمة من البيانات . فتشتمل أجهزة الكمبيوتر الشخصية الحديثة على 64-128MB من RAM (الذاكرة العشوائية) تحت تصرفها . ولا يمكن لذلك الحجم الصغير من الذاكرة أن يستوعب الأحجام الهائلة من البيانات المخزنة . ولهذا تكمن فائدة file handling فى أنه يمكنك استخدام محرك أقراص ثابتة ضخم لتخزين كميات هائلة من البيانات بصفة دائمة ويمكنك جلب أجزاء من البيانات عند الحاجة لذلك .

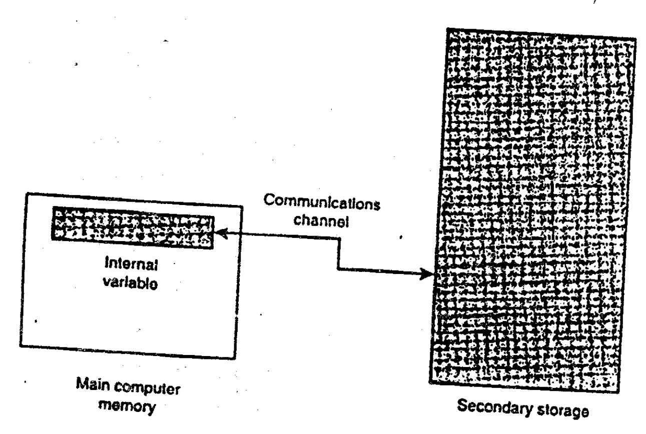


العلاقة بين الكمبيوتر و Secondary Storage.

أنواع الذاكرة:

إن التقنية الموضحة فى هذا الفصل تتمثل فى استخدام متغير واحد يحتاج لحجم صغير جدًا من ذاكرة RAM الرئيسية لحفظ عناصر من البيانات المفردة أثناء عملية المعالجة . فيتم حفظ البيانات الرئيسية فى ملف خارجي على القرص المرن . ويمكن للقرص المرن أن يكون أيضًا محرك الأقراص الثابتة أو محرك أقراص على الشبكة (network drive) أو remote server فى استراليا . ولا يعتبر ذلك أمرًا مهمًا مادام أنه ليس ذاكرة رئيسية .

والتخزين الخارجي يشتمل على كم هائل من البيانات التي لا يمكن أن تستوعبها الذاكرة الرئيسية في جهاز الكمبيوتر لأن حجمها صغير نسبيًا. فالمهم هو إنشاء صورة من صور الاتصال بين الذاكرة الرئيسية والتخزين الخارجي، ثم نقل مجموعات صغيرة من البيانات من التخزين الخارجي إلى الذاكرة الرئيسية. ويعد جهاز التخزين الخارجي جزءاً من ذاكرة الكمبيوتر الرئيسية. وأخيراً يعرف التخزين الخارجي بالـ virtual memory (الذاكرة الافتراضية).



العلاقة بين ذاكرة جهاز الكمبيوتر والذاكرة الافتراضية الموجودة في Secondary Storage.

كتابة وقراءة البيانات:

يشتمل تخزين البيانات على عمليتين أساسيتين، هما كتابة البيانات وقراءتها. وسنوضح أولاً عملية كتابة البيانات. وهي عملية مهمة جداً لأنك إذا لم تكتب بياناتك أولاً، فلن تستطيع قراءتها فيما بعد.

عملية كتابة البيانات:

تعتبر عملية كتابة ملف على قرص في لغة ++C عملية واضحة وتتمثل في الخطوات التالية:

١- قم بإعـداد اسم الملف الخـارجي الذي سيتم تخزينه في محرك الأقراص المرن :

```
char filename[20] = "a:test.txt";
```

٢- والآن قم بتـحـديد نوع الملف الذي ستعالجه. نوعه هنا ملف مخرج (output file).

```
int mode = ios::out;
```

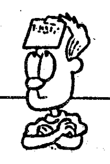
٣- قم بتحديد أسلوب التعامل مع محرك الأقراص الذي هو في هذه الحالة OOP. ويعتبر fout نموذج من fstream الذي يمكنه الوصول إلى كل عناصـر functions الموجـودة في fstream class.

```
fstream fout( filename, mode );
```

٤- قم باستخدام الكود الخاص بلغة C++ والمعـالج بأسلوب OOP لقـراءة Character من لوحـة المفاتيح وإرساله إلى الملف الخارجي باسم a:test.txt. فتقـوم عـمليـة (cin.get (ch بقراءة البيانات من لوحة المفاتيح. وبعد ذلك يتم استخدام (fout.put (ch لإرسـال تلك البيـانات إلى جهـاز Secondary storage، الذي يتـمـثـل هنا في القرص المرن.

```
while ( cin.get(ch) )
{
    fout.put( ch );
}
```

٥- وأخيراً يجب عليك إغلاق الملف، من خلال fout.close التي هي عبارة عن function موجودة في fstream class ويتم استخدامها لإنهاء الاتصال بجهاز Secondary storage.



فقد الملف

تحدث عملية فقد الكمبيوتر للملف بشكل متكرر. فعندما تقوم بحفظ بياناتك، يكون لديك خياران. يتمثل الخيار الأول في تحديد اسم الملف ثم ترك الكمبيوتر ليحدد المكان الذي سيضعه فيه. ويسمى ذلك المكان بـ default location (الموقع الافتراضي). أما الخيار الثاني فيتمثل في عملية تحديد كل من الموقع واسم الملف. فعندما تتعامل مع نظام صغير، ستجد بياناتك بسهولة ولن تكون هناك مشكلة. ولكن عند تعاملك مع نظام كبير، يمكن أن تفقد بياناتك. ولذلك يجب عليك استخدام الطريقة الثانية في تخزين الملفات ، على أن يكون القرص المرن الخاص بك هو عبارة عن موقع التخزين.

```
fout.close();
```

٦- عندما تقوم بتشغيل هذا البرنامج الموضح فى مثال (١٨-١)، يجب عليك تسجيل سلسلة متتالية من الأحرف. ولإنهاء العملية وحفظ البيانات على القرص، اضغط على زرى Ctrl و z معًا. وقد استخدمت تلك السلسلة المتتالية من الأحرف لتجنب الكود الإضافى الذى يمكن أن يصرف انتباهك عن فهم عملية الـ file handling البسيطة. كما أن نظام التشغيل يستخدم تلك السلسلة أيضًا.

ويعرض نموذج المخرجات التالى البيانات التى تم إدخالها من لوحة المفاتيح. مع ملاحظة أنه لم يتم الضغط على Ctrl+z حتى الآن.

```
Ready for input: Use Control-Z to end.
Create a file on your flop
```

مثال (١٨-١): إنشاء ملف

```
#include <fstream.h>

main()
{
   char ch;
   char filename[20] = "a:test.txt";
   int mode = ios::out;

   fstream fout( filename, mode );      // Output file
   cout << "Ready for input: Use Control-Z to end."
<< endl;
   while ( cin.get(ch) )
   {
      fout.put( ch );
   }
   fout.close();
   return(0);
}
```



عدم حفظ الملف

هناك سببان لعدم قيام الكمبيوتر بحفظ الملف. يتمثل السبب الأول فى أن الموقع الذى اعتقدت أنه الموقع المحدد لحفظ الملف لم يكن هو الموقع الصحيح. وفى هذه الحالة سيكون الملف مختفياً فى مكان ما ولكنه يكون موجوداً فعلاً. ويتمثل السبب الثانى فى أنه إذا تم فقد السطر ( ) fout.close، لن يقوم الكمبيوتر بغلق الملف، وبالتالى ستفقد البيانات للأبد.



التأكد من حفظ البيانات

إذا كنت على دراية بمبادئ IT الرئيسية، فقم بفحص محتويات القرص المرن الخاص بك. وإذا كان كل شئ سليماً، ستجد هناك ملف اسمه a:test.txt. افتح هذا الملف باستخدام Notepad (المفكرة)، وستجد به البيانات التى قمت بتسجيلها أثناء تشغيل البرنامج السابق.

علامة eof:

عندما قام البرنامج السابق بغلق الملف، فإنه وضع علامة فى نهاية هذا الملف تسمى eof (والتى تعتبر اختصاراً لعبارة End Of File). وهذه العلامة مفيدة لأن عدم وجودها يعنى عدم وجود الملف. وبما أن الأمر close مسئول عن وضع تلك العلامة فى نهاية الملف، فحذف هذا الأمر من البرنامج يؤدى إلى فقد البيانات.

عملية قراءة البيانات:

بعد إنشاء الملف، يمكن الآن قراءة محتوياته من خلال الخطوات التالية :

١- لقد قمت فى البرنامج السابق بإعداد اسم الملف الخارجى الذى تم تخزينه فى محرك الأقراص المرن. وعليك الآن استخدام نفس الكود السابق لقراءة هذا الاسم :

```
char filename[20] = "a:test.txt";
```

٢- والآن قم بتحديد نوع الملف الذى ستعالجه . ونوعه هنا ملف مدخل(input file) :

```
int mode = ios::in;
```

٣- ثم قم بتحديد أسلوب التعامل مع القرص وهو OOP . ويعتبر fin نموذج من fstream الذى يمكنه الوصول إلى كل عناصر functions الموجودة فى fstream class .

```
fstream fin( filename, mode );
```

٤- يفضل التأكد من أنه يمكنك فتح الملف واستخدامه وإذا لم تستطع ذلك، قم بإصدار رسالة تنبيه . ويجب عليك استخدام الكود التالى :

```
if (!fin)
  cerr << "Unable to open file";
```

٥- وبعد ذلك قم باستخدام الكود fin.get (ch) لقراءة characters من الملف وإرسالها إلى الشاشة :

```
while ( fin.get(ch) )
    {
          cout << ch;
    }
```

٦- ومادام fin.get (ch) يقوم بقراءة char، فإنه يساعد loop على الاستمرار فى العمل . وعندما يصل إلى eof، تتوقف loop .

٧- وأخيراً قم بإغلاق الملف :

```
fin.close();
```

تأكد من أن القرص المرن الخاص بك موجود فى موضعه وحاول أن تنفذ البرنامج الموضح فى مثال (١٨-٢) . وعندما تفعل ذلك، ستجد الرسالة الأصلية التى سجلتها أثناء تشغيل البرنامج الأول . شاهد الإخراج التالى وتأكد من أن الملف الأصلى يعمل بالفعل :

```
Create a file on your floppy disk
```

مثال (١٨-٢): قراءة ملف

```
#include <fstream.h>

main()
```

```
{
   char ch;
   char filename[20] = "a:test.txt";
   int mode = ios::in;

   fstream fin( filename, mode );     // Input file
   if (!fin)
      cout << "Unable to open file";
   while ( fin.get(ch) )
   {
      cout << ch;
   }
   fin.close();
   return(0);
}
```

استعراض Text Files و Binary Files:

لقد لاحظت حتى الآن أن لغة ++C تتعامل مع text files . وتقوم text files بتخزين البيانات باستخدام أسلوب ASCII وتحتاج فعلاً لمعالجة أخرى عند استخدامها فى البرامج الرقمية الكبيرة. ومع ذلك يمكنك تجاهل هذا النوع من الملفات واستخدام binary files . وتقوم binary files بتخزين أى نوع من البيانات وتعتبر أفضل من text files . وستوضح البرامج التالية ذلك.

كيفية كتابة Binary Files:

قم بتحديد اسم الملف بنفس الطريقة السابقة:

```
char filename[20] = "a:xtest.dat";
```

قم بعد ذلك بإعداد نوع الملف الذى ستستخدمه. ونوعه هنا ملف مخرج. ولكن لاحظ كيفية إنشاء binary file :

```
int mode = (ios::out | ios::binary);
```

بعد تحديد نوع الملف، عليك الآن التعامل مع الملف كما سبق:

```
fstream fout( filename, mode );
```

وبعد ذلك، قم ببساطة بإرسال البيانات الموجـــودة فى array إلى الملف الخـــارجى باستخدام for loop. ولاحظ أنه يجب عليك استخدام end l بعد كل عنصر يتم إرساله إلى الملف. ويعـرف ذلك بالـ delimiter (الحـرف المحـدد) الذى يقـوم بفصل كل قيمة من نوع integer من النص. وبدونه لا يمكنك استعادة البيانات الأصلية بسهولة:

```
for (loop = 0; loop < MAX; loop++)
{
    fout << x[loop] << endl;
}
```

وأخيـراً قم بإغلاق الملف بسطر الكود التالى:

```
fout.close();
```



رسالة تنبيه

إذا كـان برنامـجك يعـمل بطريقـة سليمة، فلن ترى رسالة تنبيه. ولتتأكد من أن تلك الرسـالة تعـمل، عليك أن تقوم بالتالى: عندما يعرض البرنامج اسم الملف "a:test.txt"، قم بتـغيير هذا الاسم إلى أى اسم آخر مختلف مثل "a:wrong.txt". وفى هذه الحالة عندما تقوم بتشغيل البرنامج، فإنه لن يجد الملف وسيقدم هذه الرسالة.

مما سبق يمكنك ملاحظة أن طريقة إنشاء binary files مطابقة لطريقة إنشاء text files. ويكمن الاختلاف الوحيد فى أنك تقوم بإعداد الملف فى صورة binary file من البداية. حاول أن تنفذ البرنامج الموجود فى مثال (١٨-٣) لإنشاء الملف الخارجى على القرص. ولاحظ أنه ستظهر القليل من المعلومات على الشاشة. والعلامة الوحيدة التى ستظهر على الشاشة هى الرسالة القصيرة التى تقول Data written to file. وعلى الرغم من ذلك يقرأ المستخدمون هذه الرسالة كـ Your program didn't work. ولكن لا داع للقلق لأن رسالة Data written to file تؤكد أن كل شئ يعمل بطريقة سليمة.

مثال (١٨-٢): إنشاء Binary File

```
#include <fstream.h>
#define MAX 6

main()
{
    int loop;
    int x[MAX] = {42,707,99,101};
```

```
    char filename[20] = "a:xtest.dat";
    int mode = (ios::out | ios::binary);

    fstream fout( filename, mode );     // Output file
    for (loop = 0; loop < MAX; loop++)
    {
       fout << x[loop] << endl;
    }
    fout.close();
    cout << "Data written to file" << endl;
    return(0);
}
```

كيفية قراءة Binary Files:

سنقوم الآن بكتابة برنامج لقراءة محتويات binary file . ويعتبر ذلك مطابق أيضاً لمثال (١٨-٢) "نسخة text file" . والاختلاف الوحيد الموجود فى مثال (١٨-٤) يكمن فى السطر الذى يحدد نوع الملف المستخدم . وبالطبع نوع الملف هنا ملف مدخل يستخدم أسلوب binary .

```
int mode = (ios::in | ios::binary);
```

حاول أن تنفذ البرنامج الآن ولاحظ الأرقام التى ستظهر مرة ثانية على الشاشة . وبما أن البيانات الحقيقة هى التى ستظهر على الشاشة ، فلن تحتاج لرسالة تطمأنك أن كل شئ يعمل بطريقة صحيحة .

```
42
707
99
101
0
0
```

مثال (١٨-٤): قراءة Binary File

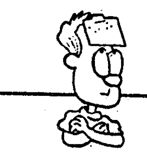
```
#include <fstream.h>

main()
{
   int loop = 0;
```

```
    int x;
    char filename[20] = "a:xtest.dat";
    int mode = (ios::in | ios::binary);

    fstream fin( filename, mode );     // input file
    if (!fin)
       cerr << "Unable to open file";
    while (fin >> x)
    {
       cout << x << endl;
       loop++;
    }
    fin.close();
    return(0);
}
```

إنشاء Compound Files (الملفات المركبة):



تعديل الملفات

تستخدم binary files السابقة قيم من نوع integer. حاول تعديل هذه الملفات لتستخدم floats. وإذا نجحت في ذلك، حاول أن تستخدم char.

سنشرح الآن كيفية تخزين واسترجاع ++C structures التى استعرضناها فى الفصل الرابع عشر "معالجة أنواع مختلطة من البيانات باستخدام Structure" . وتستخدم structures فى تمثيل السجلات ذات البيانات التطبيقية، وتتكون من مجموعات من البيانات المترابطة والتى من أنواع بيانات مختلفة . والجدير بالذكر أن طريقة الوصول للملف هنا مطابقة للطريقة التى تم استخدامها مع binary files . ولكن يجب استخدام معاملات الإدخال والإخراج التى تم شرحها فى الفصل السادس عشر "عرض عملية تحميل معاملات التشغيل" لتتلائم مع طبيعة الملفات المستخدمة هنا . اتبع الخطوات التالية :

١ - قم بتحديد structure بالطريقة العادية الموضحة فى الفصل الرابع عشر .

٢ - قم بتحميل معامل الإخراج لتتعامل مع هذا الـ structure، كما هو موضح فى الفصل السادس عشر .

٣- قم بإعداد array تشتمل على المعلومات المطلوبة ، كما هو موضح فى الفصل الخامس عشر "استخدام Structures فى عمليات الإدخال والإخراج" .

٤- قم بتحديد نوع الملف ثم افتح binary file كما سبق .

٥- استخدام for loop لإرسال البيانات إلى الملف الخارجى .

٦- قم بإغلاق الملف .

شغل البرنامج الموجود فى مثال (١٨-٥) ولاحظ الضوء الذى سيظهر على محرك الأقراص الخاص بك . ويعتبر ذلك الضوء دليل على بدء العمل . ولم يتم ذكر الإخراج هنا لأنه يتمثل فقط فى رسالة Data written to file .

مثال (١٨-٥): إنشاء ملف السجلات

```
#include <fstream.h>
struct planet     {
            char name[10];
            int  dist;
          };

ostream& operator << (ostream& str_out, planet& d)
{
   str_out << d.name  << endl;
   str_out << d.dist << endl;
   return(str_out);
};

main()
{
struct planet solar[3] = {    {"MERCURY",58},
                              {"VENUS",108},
                              {"EARTH",150}};
   int x;
   char filename[20] = "a:system.dat";
   int mode = (ios::out | ios::binary);

   fstream fout( filename, mode );      // Output file
   for (x = 0; x < 3; x++)
   {
```

```
    fout << solar[x];
    }
    fout.close();
    cout << "Data written to file" << endl;
    return(0);
}
```

كيفية قراءة الملفات المركبة:

سنقوم الآن بكتابة برنامج لقراءة محتويات binary file المركب. ويعتبر م (١٨-٦) مطابق تقريباً لمثال (١٨-٢) فيما عدا أنه يجب تحميل معاملات الإدخ والإخراج لمعالجة structures ويجب استخدام طريقة الوصول إلى binary file .

١- بعد تحميل معاملات الإخراج والإدخال، قم بإعداد أسلوب التعامل لمعالجة الملف الخارج

٢- افتح الملف واقرأ البيانات، باستخدام معامل الإدخال الذى تم تحميله.

٣- قم بإغلاق الملف.

وسيظهر الإخراج على الشاشة كمايلى:

```
MERCURY  58
VENUS    108
EARTH    150
```



فائدة المعاملات المحملة

يمكن تنسيق البيانات فى معامل محمل ثم استخدامه عند الحاجة، بدون إضافة أى كود آخر. ويمكن ملاحظة ذلك من خلال معامل الإخراج المحمل الذى تم استخدامه هنا.

مثال (١٨-٦): قراءة ملف السجلات

```
#include <fstream.h>
#define tab '\t'
```

```
struct planet    {
                char name[10];
                int  dist;
               };

ostream& operator << (ostream& str_out, planet& d)
{
   str_out << d.name << tab;
   str_out << d.dist << tab << endl;
   return(str_out);
};

istream& operator >> (istream& str_in, planet& d)
{
   str_in >> d.name;
   str_in >> d.dist;
   return(str_in);
};

main()
{
   struct planet solar;
   char filename[20] = "a:system.dat";
   int mode = (ios::in | ios::binary);

   fstream fin( filename, mode );  // Input file
   if (!fin)
      cerr << "Unable to open file";
   while (fin >> solar)
   {
      cout << solar;
   }
   fin.close();
   return(0);
}
```

إنشاء برامج:

قم بإنشاء برنامج يكون قادراً على تخزين المعلومات التالية الخاصة بالنظام الشمسى باستخدام Secondary storage . ولا تقم باستدعاء ملف البيانات الخاص بك المسمى system.dat . ولكنك ستحتاجه فى الفصل القادم .

PLANET	*DISTANCE*	*MOONS*	*YEAR*
MERCURY	58	0	0.240
VENUS	108	0	0.625
EARTH	150	1	1.000
MARS	228	2	1.910
JUPITER	778	12	12
SATURN	1427	14	29.9
URANUS	2869	5	85.24
NEPTUNE	4498	2	167.19
PLUTO	5900	1	251.29

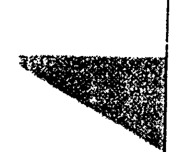
قم بإنشاء برنامج آخر لقراءة وعرض المعلومات السابقة .

أهم الدروس المستفادة:

- يعتبر File handling وسيلة لتخزين البيانات إلكترونيًا على جهاز خارجى متصل بالكمبيوتر مثل محرك الأقراص، وذلك لاستخدامها مرة ثانية . ولا تختفى تلك البيانات عند إغلاق جهاز الكمبيوتر، بل تبقى مخزنة .
- تحتاج الكميات الضخمة من البيانات إلى التخزين، ولكن إمكانيات ذاكرة الكمبيوتر محدودة . فيمكن File handling الكمبيوتر من تسجيل كميات صغيرة من تلك البيانات ومعالجتها ثم حذفها .
- يمكن File handling الكمبيوتر من استخدام سعة Secondary storage والتعامل معها كجزء من ذاكرته تعرف باسم الذاكـــرة الإضافيـــة (virtual memory) .
- تقوم text files بتخزين البيانات باستخدام نظام ASCII البسيط .
- تقوم Binary files بتخزين البيانات سواء كانت رقمية أو مركبة بصورة يمكن من خلالها الوصول بسهولة إلى تلك البيانات ومعالجتها .
- يتم إدراج علامة eof باستخدام الأمر close لتحديد نهاية الملف عند كتابته فى Secondary storage .


الفصل الثالث عشر
فيجوال سي ++

الفصل الثالث عشر
فيجوال سى ++



لأننا نعتقد أن أفضل طريقة لتعلم البرمجة هي كتابة البرامج فدعنا فيما يلي نحاول إطلاعك على كل ما يتعلق بكتابة Code لبرنامج في لغة C باستخدام مترجم Visual C++ خطوة بعد أخرى ، وندعم ذلك بأمثلة توضيحية قد تجدها بسيطة في البداية لكنها سرعان ما ستتدرج إلى مستويات أعمق ، وسنبدأ التعرف على طريقة كتابة البرامج من خلال المثال التالي الذي سنوضح من خلاله طريقة الاستفادة من أستوديو المطورين الذي يعد ميزة أساسية في Visual C++ .

أستوديو المطورين هو تطبيق متكامل يمثل واجهة سهلة الاستخدام للوصول من خلالها إلى أدوات البرمجة القوية التي توفرها Visual C++

- فيمكنك استخدامه لإدارة مشروع البرنامج .
- إنشاء وتعديل ملفات برامج المصدر .
- تصميم مصادر البرنامج مثل القوائم وصناديق الحوار والرموز ... الخ.
- ويمكنك أيضاً استخدام أستوديو المطورين لإنتاج كود لمكونات البرنامج الأساسية من خلال المعالجات السحرية Wizards التي تتوافر مع Visual C++ .
- وأيضاً يمكنك بناء وترجمة Compiling وتصحيح Debug أخطاء البرامج.

ونحن من خلال هذا الفصل لا نسعى إلى شرح Visual C++ بقدر ما نهدف إلى تعريفك بواجهة العمل حتى تستطيع تنفيذ الأمثلة التي سنشرح من خلالها لغة C القياسية ولغة C++ قبل الخوض في تفاصيل Visual C++ .

تشغيل أستوديو المطورين

عندما تقوم بتثبيت Visual C++ فإن برنامج التثبيت ينشئ رمز - Icon خاص بهذا التطبيق داخل المجموعة Visual C++ التي تتواجد في قائمة بدء التشغيل ولتشغيل أستوديو المطورين Developers Studio يكفي أن تستخدم الأمر:

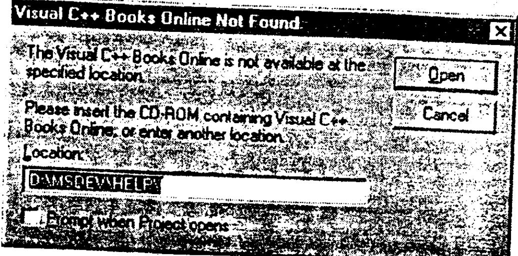
Start\Programs \ VisualC++ \ Developers Studio

وبمجرد بدء تشغيل البرنامج يبحث عن مجموعة ملفات المساعدة التي يستخدمها كمرجع للمستخدم أثناء العمل ، والتي تكون - كما هو الحال في الغالب - موجودة على القرص المدمج CD-ROM ما لم تكن وضعتها على القرص الصلب أثناء عملية تثبيت البرنامج .



إذا لم تكن قمت بتثبيت البرنامج على القرص الصلب في جهازك راجع كيفية التثبيت من خلال الملحق الموجود في نهاية الكتاب .

فإن لم يجد البرنامج القرص الصحيح في موضعه يظهر الرسالة التالية لتضع له القرص في المشغل وتضغط زر open لفتحه أو تلغي عملية الاستعانة بمجموعة الكتب الفورية Books on Line أثناء العمل بضغط زر Cancel.



ومجموعة الكتب الفورية هذه هي المحتوية على شرح تفصيلي لمكتبات اللغة والدوال المبنية فيها وطريقة تعاملها مع المتغيرات والثوابت وتوضح في هذه الصورة الإلكترونية لتسهيل عمليات البحث من خلالها عن أي معلومات مطلوبة كما أنها تكون متاحة لك دائماً بمجرد الوقوف بمؤشر الكتابة عند الكلمة التي تحتاج إلى شرح لها وضغط مفتاح F1 .

وسيلي في نهاية الفصل شرح كيفية الحصول على المساعدة أثناء العمل ، وطريقة الاستفادة من الكتب الفورية .

بصفة عامة يتكون البرنامج في Vasual C++ وفقاً لعدة خطوات محددة سنتناول بالترتيب كما يلي:

1 - إنشاء المشروع Project

بمجرد بدء أستوديو المطورين فإن أول ما يجب عليك عمله هو إنشاء المشروع الذي سيتم من خلاله متابعة وإدارة العمل في البرنامج الذي ستكتبه ، والمشروع في Visual C++ يقوم بحفظ جميع المعلومات المطلوبة لتكوين برنامج معين ، وتتضمن هذه المعلومات أسماء ملفات المصدر Source Files الخاصة بالبرنامج ،

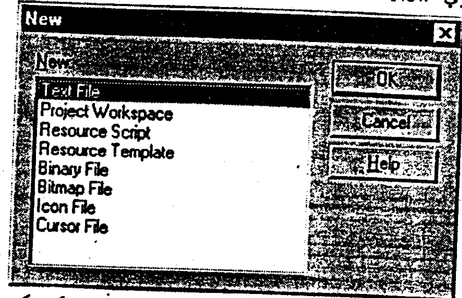
والعلاقات بينها ، وقائمة بملفات المكتبات المطلوبة ، وقائمة بجميع خيارات تشغيل المترجم – Compiler والرابط Linker ... وخيارات تشغيل الأدوات الأخرى المستخدمة في البرنامج.

وفيما يلي سنقوم بتكوين مشروع بسيط الغرض منه التعرف على واجهة العمل وسنطلق عليه اسم chapter1 ولكي تنشئ المشروع الخاص به تابع الخطوات التالية:

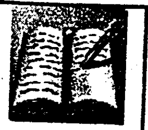
1 – من خلال أستوديو المطورين استخدم الأمر

File\New

لفتح الصندوق الحواري New

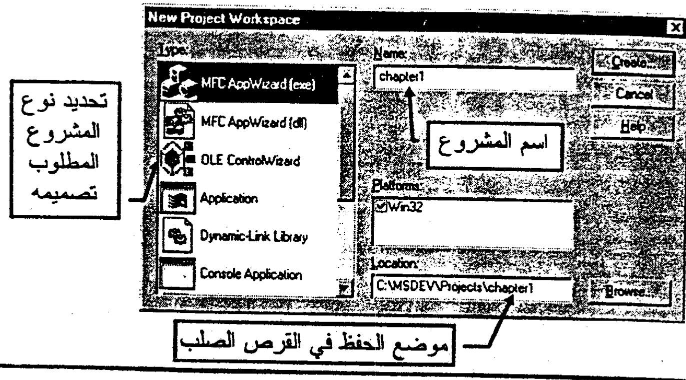


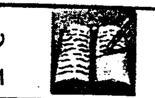
2 – ستلاحظ من خلال الصندوق أن هذا الأمر يقوم بإنشاء أي ملف جديد يمكن تكوينه من خلال أستوديو المطورين، ولذلك فمن القائمة حدد الخيار Project workspace لتكوين مساحة المشروع ثم اضغط (موافق – OK).



يتيح أستوديو المطورين تكوين أكثر من مشروع داخل Project workspace في وقت واحد لكن من خلال هذا الكتاب سنتعامل في كل مرة مع مشروع واحد ... ويمكنك معرفة المزيد عن هذه النقطة من خلال مراجعة المساعدة الفورية Books on line Help في موضوع Visual C++ Books \User's Guide \ Visual C++ User's Guide\ Working With Projects

3 – اضغط (موافق – OK)يظهر الصندوق الحواري التالي New Project workspace .





راجع أنواع المشروعات في الملحق الخاص بها في نهاية الكتاب.

4 - من الصندوق الجديد وفي مربع النص :Name ادخل اسم المشروع chapter1 حيث سيكون هذا الاسم هو اسم المشروع أو البرنامج الذي تقوم بإنشائه.

5 - في مربع النص :Location حدد المجلد- Foider الذي تود إنشاء أو حفظ المشروع به وستجد أن البرنامج يقترح له نفس اسم المشروع وسيضعه داخل المجلد C:\MSDEV\Project وهو المجلد الافتراضي الذي يحفظ فيه البرنامج المشاريع الجديدة ، فإن أردت التغيير حدد المجلد المطلوب وإن لم يكن موجوداً بالفعل على القرص الصلب يستطيع أستوديو المطورين أن ينشئه.

6 - في القائمة :Type حدد النوع Console لإنشاء تطبيق من النوع 32-bit Character mode ليعمل في بيئة DOS .



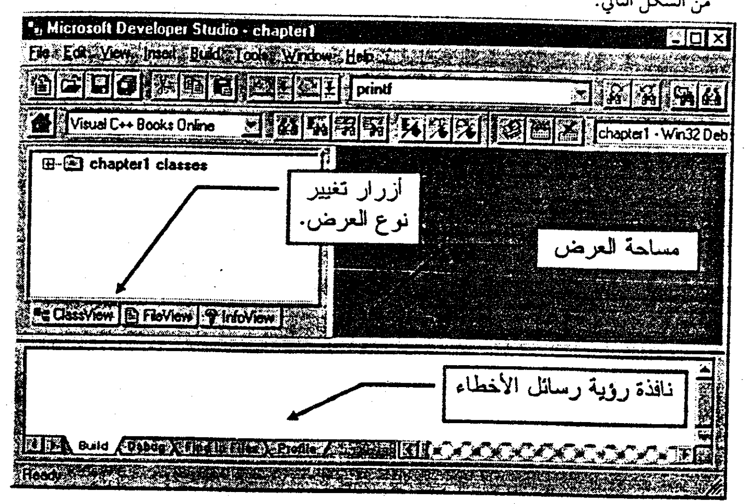
التطبيقات من النوع Console يمكن التعامل معها من خلال نافذة أو في حالة شاشة كاملة مثل MS-DOS .

7- من خلال نفس الصندوق الحواري تأكد من تحديد الخيار Win32 في المنطقة Platforms وغالباً ستجد أنه الخيار الوحيد ، وتحديد هذا الخيار يعني أن البرنامج الجديد سيكون مدعوماً بـ win32 ، يعني أنه يستطيع استدعاء وظائف win32 التي تمثل مجموعة API لكل من Windows95 و Windows NT.



API هـي اختصـار **Application Programming Interface** الـتي تمثـل مجموعـة الوظـائف الأساسـية لواجهـة المستخدم ، وأسلوب تعامل البرنامج مع نظام التشغيل.

8- اضغط زر (إنشاء – Create) في الصندوق الحواري وبذلك ينشئ أستوديو المطورين مساحة مشروع جديـدة ...ويسميه Chapter1 وهو يحتوي على ملف وحيد أيضاً يسمى Chapter1 كما يظهـر من الشكل التالي:

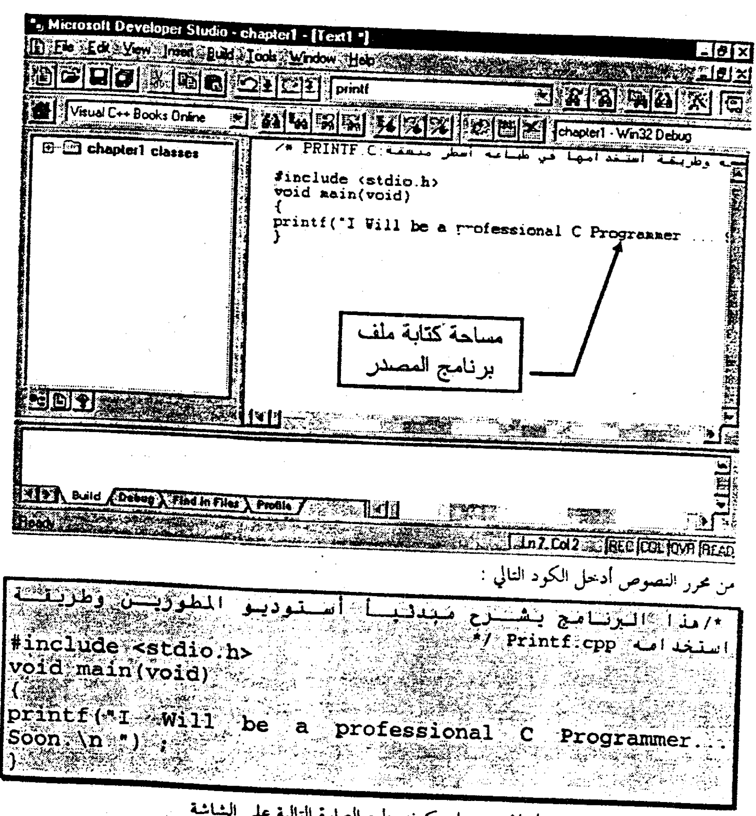


2 - إنشاء وتعديل ملف المصدر

الخطوة التالية في تكوين البرنامج هي إنشاء ملف المصدر Source File للبرنامج Chapter1.cpp ولكي تفعل ذلك استخدم الأمر

File \ New

ومن الصندوق الحواري حدد الخيار Text File وبعدها اضغط OK لفتح معالج نصوص يسمح لك بكتابة Source File للبرنامج كما يلي:



من محرر النصوص أدخل الكود التالي :

```
/*هذا البرنامج يشرح مبدئيا أستوديو المطورين وطريقة استخدامه Printf.cpp */
#include <stdio.h>
void main(void)
{
printf("I Will be a professional C Programmer...
Soon.\n ") ;
}
```

وهذا البرنامج ذو مهمة بسيطة لا تزيد على كونه يطبع العبارة التالية على الشاشة

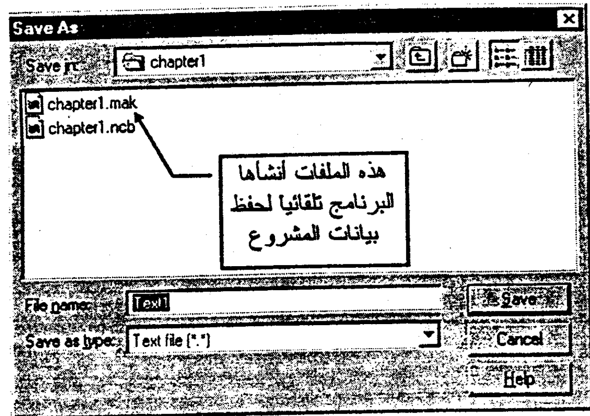
I Will be a professional C Programmer... Soon.

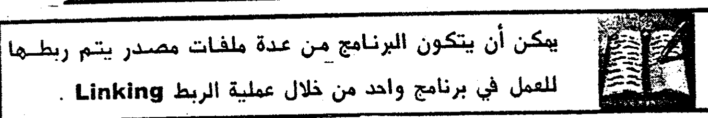
باستخدام دالة موجودة في مكتبات C وهي دالة `printf( )` .

بمجرد الانتهاء اضغط الأمر

File\Save or Save As

لحفظ الملف أو اضغط الرمز 💾 من شريط الأدوات وتأكد من أنك تحفظ الملف في نفس الفهرس الذي حفظت به المشروع حتى نتمكن بعد ذلك من ترجمة الملف . وستجد أن البرنامج يقترح للملف اسم Text1 وافق على الاسم أو غيره كما شئت ثم اضغط حفظ SAVE كما بالشكل التالي :





يمكن أن يتكون البرنامج من عدة ملفات مصدر يتم ربطها للعمل في برنامج واحد من خلال عملية الربط **Linking** .

والآن دعنا نأخذ خطوة أخرى في بناء البرنامج ونضيف الملف المصدر إلى المشروع

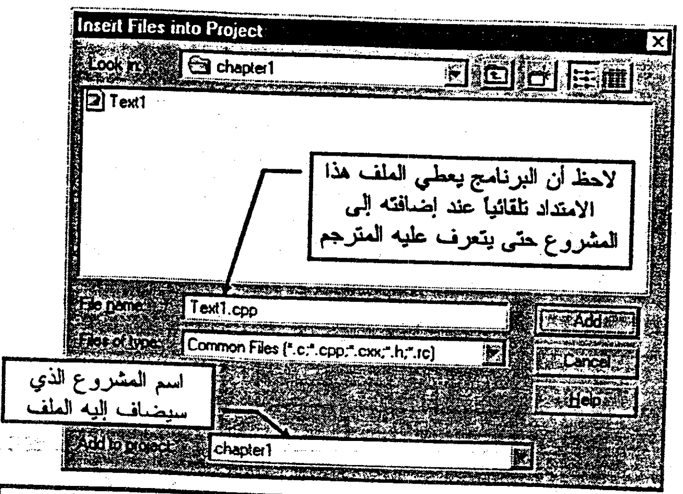
3 - إضافة ملف المصدر للمشروع

بعد إنشاء وحفظ ملف المصدر للبرنامج text1 يجب إضافته إلى المشروع chapter1 ولكي تفعل ذلك استخدم الأمر التالي:

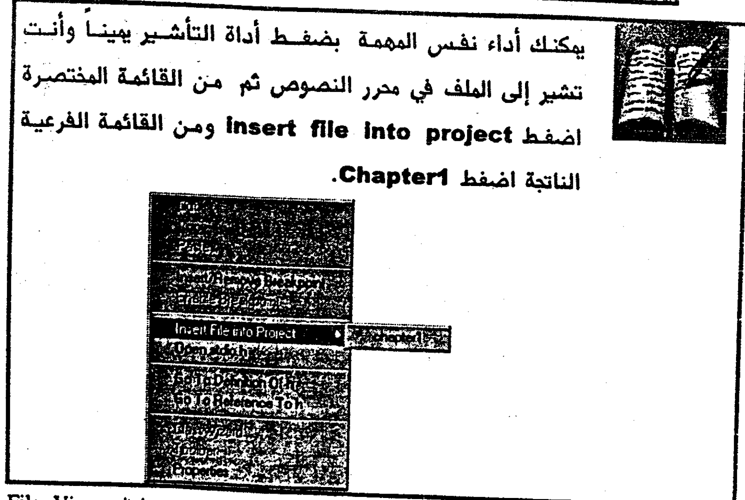
من النافذة أستوديو المطورين اضغط الأمر

Insert\File into Project

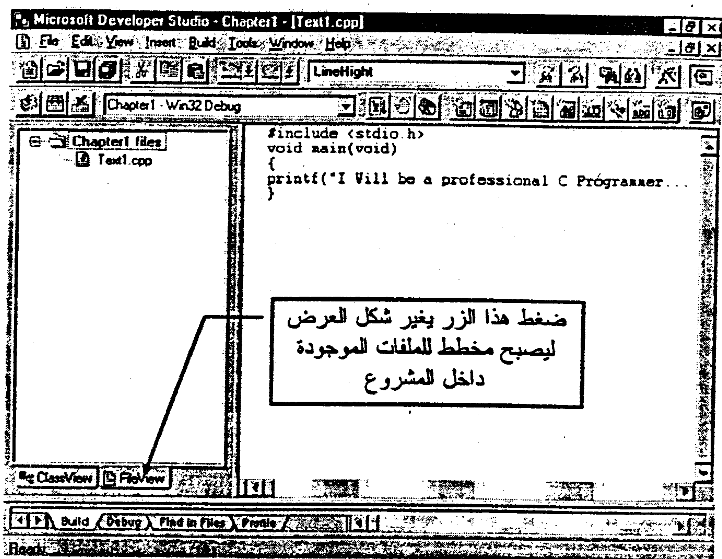
ومن الصندوق الحواري الناتج حدد الملف الذي ترغب في إضافته إلى فهرس المشروع كما بالشكل التالي ثم اضغط (إضافة - Add)



يمكنك أداء نفس المهمة بضغط أداة التأشير يميناً وأنت تشير إلى الملف في محرر النصوص ثم من القائمة المختصرة اضغط **insert file into project** ومن القائمة الفرعية الناتجة اضغط **Chapter1**.



وبعد إضافة الملف للمشروع يظهر في شكل **File View** والذي تستطيع إظهاره بضغط الزر **File View** في نافذة أستوديو المطورين.



وبذلك يظهر اسم المشروع Chapter 1 وملفات المصدر التي تنتمي إليه وهو في هذه الحالة الملف الوحيـــد الذي أضفناه.

وستجد إلى جوار كل فرع رئيسي في المخطط تظهر علامة (+) فإن ضغطتها بأداة التأشير يتم تمديد المخطط لعرض محتويات كل ملف مصدر من:

- التصنيفات Classes وسيأتي شرحها في الفصل الحادي عشر.
- الدوال Functions وسيأتي شرحها في الفصل الرابع .

ويوضع أمام كل منها رمز مميز لها بحيث عند ضغط هذا الرمز في المخطط يظهر الكود الخاص به في الجهـــة اليمنى وهو ما يسهل الوصول إلى أي جزء من الكود لفحصه أو تعديله.

وتستمر عملية التمديد بضغط علامات (+) أمام كل فرع لرؤية محتوياته حتى تظهر علامة (-) فـــإن أردت إغلاق أحد الأفرع تضغط علامة (-) لعودته إلى حالته الأولى كما بالشكل.

Chapter1 files

وكما سنرى بعد ذلك مع المشروعات الكبيرة التي تحتوي على عدة ملفات مصدر يكون هذا المخطط نافع للغاية ولاحظ أنه يمكنك فتح أي ملف مصدر أو إظهاره في مساحة الرؤية (إذا كان مفتوحاً) بالضغط مزدوجاً على الرمز الخاص به في المخطط.

وبخلاف File view ستجد أن هناك نوعين آخرين من العرض .

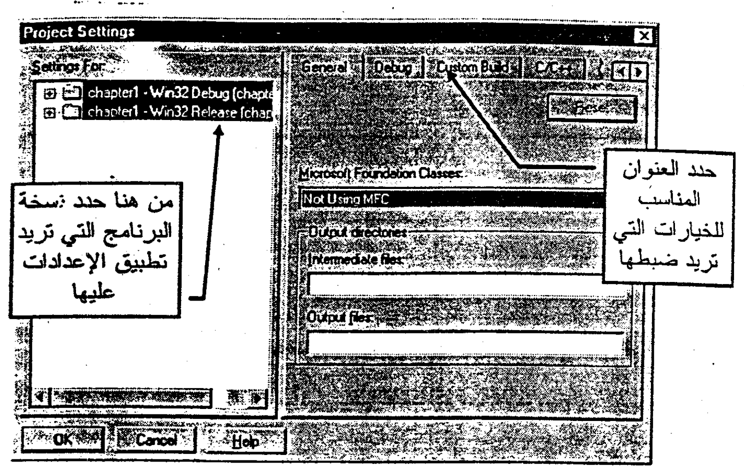
- Class view ويعرض مخطط بنفس الطريقة للتصنيفات الموجودة في المشروع وسيأتي مناقشته في الفصل الحادي عشر كما ذكرنا من قبل.
- Info View والذي يعرض مخطط للمعلومات المساعدة المتوفرة في مجموعة الكتب الفورية مقسمة بنفس الأسلوب التخطيطي ، وهو يكون متاحاً حتى ولو لم تفتح مشروعاً معيناً بشرط تواجد قرص CD في المشغل عند فتح أستوديو المطورين.
- وستجد أن الموضوعات الرئيسية يظهر إلى جوارها رمز كتاب في حين أن الموضوعات الفرعية تظهر إلى جوارها رمز صفحة بيانات للدلالة على أنها ليس لها فروع.
- وللمزيد من المعلومات حول طريقة التعامل مع Info View راجع المساعدة الفورية تحت العنوان Finding Information .

4 - ضبط إعدادات البناء

قبل البدء في استكمال خطوات تكوين البرنامج بالترجمة والربط توجد مجموعة من الخيارات التي تحتاج لضبطها للحصول على ما تحتاجه بالضبط من عملية الترجمة والربط ويمكنك التحكم فيها من خلال الأمر

Build \ Settings

لفتح الصندوق الحواري التالي



وستجد من خلال الصندوق الحواري القائمة Settings for تحتوي على صورتين لتهيئة المشــــروع كــل منهما تنتج نسخة معينة من البرنامج.

الأولى مسماة win32 Debug - chapter1 التي تنتج نسخة التصحيح Debugging من البرنامج والتي تعمل عليها حتى تصل إلى الشكل النهائي للبرنامج .

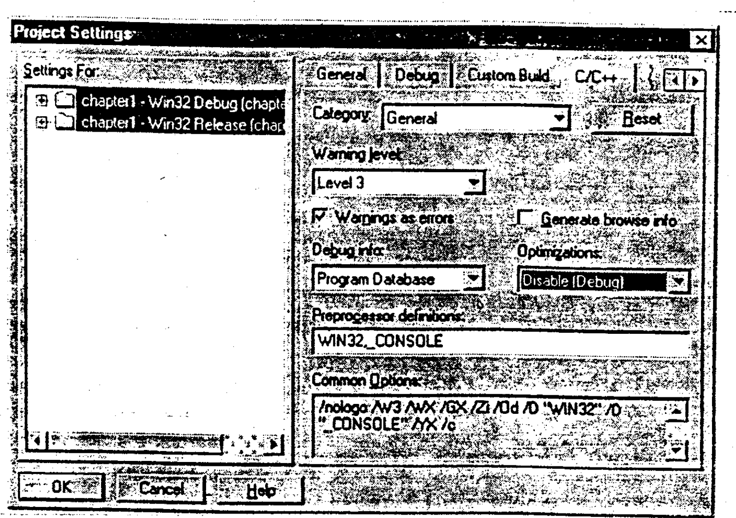
الثانية مسماة chapter1 -win32 Release التي تستخدم لإنتاج النسخة Release النهائية للبرنامج.

وللتدريب على تغيير الإعدادات حدد النسخة chapter1 - win32 Debug حتى تكون التغـــيرات الـــتي ستحدثها مؤثرة على معالجة الملفات فقط أثناء بناءك لنسخة التصحيح من برنامجك .

■ اضغط العنوان C\C++ لإتاحة الخيارات الخاصة به ومن بينها خيارات المترجم .

■ تأكد من أن الخيار General في القائمة Category هو الخيار الحالي.

■ حدد الخيار Warning As Errors حيث يعني تحديد هذا الخيار جعل المـــترجم Compiler يظــهر رسائل تحذيرية عند وجود خطأ ، وبذلك يصبح الصندوق كما بالشكل التالي.



اضغط الزر OK في أسفل الصندوق لحفظ هذه الإعدادات .



الصندوق السابق من أهم الصناديق الحوارية في **Visual C++** وللحصول على المزيد من المعلومات عن طرق وخيارات ضبط الإعدادات لظهر الصندوق الحواري ثم اضغط العنوان الذي تريده وفي نفس العنوان اضغط زر المساعدة **Help** لفتح نافذة تشرح خيارات هذا العنوان.

فعلى سبيل المثال لو أنك تحتاج مساعدة حول معنى القائمة **Category** في العنوان **C\C++** فعند ظهور الصندوق الحواري وضغط العنوان **C\C++** تضغط زر **Help** في أسفل الصندوق لعرض المساعدة عن خيارات هذا العنوان.

5 - بناء البرنامج

الخطوة التالية التي يجب عملها - بعد الانتهاء من كتابة ملفات المصدر وحفظها في المشروع وضبط خيارات الترجمة والربط - هي البدء في تكوين الملف القابل للتنفيذ (EXE) وذلك بعمل مرحلتين

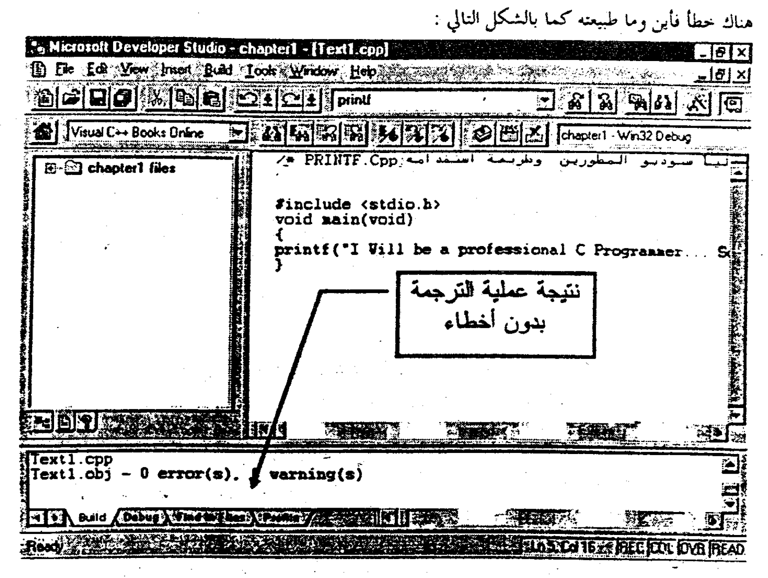
1- ترجمة الملفات إلى لغة الآلة Compiling

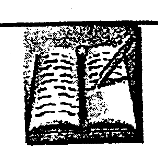
2- الربط مع المكتبات التي استعنت بها لكي تنتج الملف القابل للتنفيذ .

وتتم الخطوة الأولى من نافذة أستوديو المطورين بضغط الأمر

Build \ Compile text1.cpp   Ctrl + f7

لكي يبدأ البرنامج في ترجمة الملف ويظهر النتيجة في المساحة السفلية هل يوجد أي خطا أم لا ، ولو كان هناك خطأ فأين وما طبيعته كما بالشكل التالي :





عند وجود أي خطأ يظهر البرنامج اسم ملف المصدر الذي يحتوي على الخطأ ، رقم السطر ، رقم كود يدل على نوع الخطأ فإن حددت رقم الكود وضغطت F1 تظهر شاشة للمساعدة على حل هذه المشكلة .

والجدير بالذكر أن الأخطاء التي يتم اكتشافها بواسطة المترجم في هذه المرحلة هي الأخطاء الممكن تشبيهها بالأخطاء النحوية في اللغة البشرية مثل فتح قوس) وعدم إغلاقه بقوس (.
أو عدم إنهاء جملة بفاصلة منقوطة أو استخدام اسم معرف لم يعلن عنه كما سنعرف عند دراسة المتغيرات وهكذا ....لكن لا يمكن للمترجم أن يكتشف الأخطاء المنطقية في بناء البرنامج مادامت صيغتها من الناحية النحوية Syntax صحيحة ويتم اكتشاف الأخطاء المنطقية عن طريق عملية أخرى تعرف باسم التصحيح debugging وسيلي مناقشتها فيما بعد.

بعد انتهاء عملية الترجمة وتصحيح الأخطاء إن وجدت تبدأ عملية البناء بضغط الأمر

Build \ Build Chapter1 F7

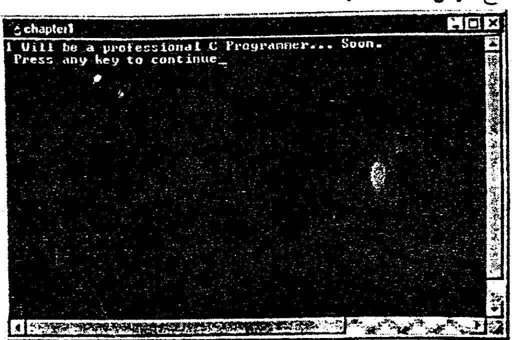


لاحظ أن القوائم يتم تعديلها تلقائياً لتناسب خياراتها أسماء الملفات التي تحددها أنت عند حفظ المشروع أو الملف .

بمجرد الانتهاء يمكنك تشغيل البرنامج بواسطة الأمر

Build\ Execute chapter1.exe

وستجد أن البرنامج يعرض الشاشة التالية :



نظرة أكثر وضوحاً على كتابة الكود

من خلال المثال السابق تعرفنا على واجهة العمل وطريقة تكوين البرنامج في Visual C++ وقبل أن ننتقل إلى الفصل التالي دعنا نتعرف على بعض المفاهيم الأساسية الهامة التي وردت في البرنامج السابق والتي سنحتاج إليها في الفصول القادمة .

أولها أنه يمكنك كتابة أية تعليقات إلى جوار الجمل الموجودة بالبرنامج دون أي تأثير لها على عمليات الترجمة والربط بشرط أن تكون هذه التعليقات بين العلامات /* نص التعليق */

ويتعرف المترجم على بدء التعليق بوجود الرمزين /* ثم نهاية التعليق بالرمزين */ وبذلك يهمل كل ما بينهما في عملية الترجمة.

ويمكن للتعليق أن يستمر ويحتل أكثر من سطر ويفضل دائماً أن تضع كم كافي من التعليقات إلى جوار براجحك لتذكيرك بكل خطوة من البرنامج وطريقة تنفيذها والهدف منها مع ملاحظة أنه

- يمكنك كتابة التعليق كما في المثال السابق في بداية البرنامج .
- يمكنك كتابة تعليق في نفس السطر مع جمل البرنامج مثل.

```
printf("hello\n") ; /* طباعة كلمة ترحيب */
```

- يمكن كتابة تعليقات في بداية نص كل دالة لشرح وظيفتها وطريقة تشغيلها.
- لا يمكن تداخل التعليقات فعلى سبيل المثال الشكل التالي ينتج خطأ في عملية الترجمة

```
/* ..........
     /* open file */
     ............
     ...........
     ...........
*/..........
```

- يمكنك استخدام العلامتين // في بداية السطر ليهمل المترجم السطر حتى نهايته لكن لن تتمكن من استكمال التعليق في السطر التالي إلا بوضع علامتين جديدتين مثل

```
// this cod is to print the report
// the operation is done by random Access
```



استخدام الرمزين // بدأ استخدامه مع C++

ثانياً السطور التالية يمكن اعتبارها برنامج مصدر كامل في لغة C .

```
void main(void)
{
printf  ("I  Will be a professional C Programmer ...Soon
");
}
```

وبداية لا حظ الاسم **( ) main** فإن كل برامج لغة C يجب أن تحتوي أساساً على الدالة () main .

والدالة ()main هي الدالة الرئيسية التي يتم نقل التحكم إليها من نظام التشغيل عند بدء تشغيل البرنـــامج ومنها ينتقل التحكم بعد ذلك إلى أي دالة فرعية أثناء التنفيذ . وهي بالطبع أول دالة يتم تنفيذها في البرنامج وهي أيضاً التي تحدد بداية البرنامج ونهايته إذا يبدأ ببدايتها وينتهي بنهايتها.

وتتكون البرامج في لغة C أساساً من هذه الدالة مع أي دوال فرعية أخرى والدالة مثل البرنـــامج الفرعـــي subroutine في لغة Basic وستتناول الدوال بالتفصيل فيما بعد لكن الآن يجب معرفة أن ()main هـــي الدالة الأساسية في برامج C وأن أي برنامج في لغة C يتكون منها مع احتمال وجود دالة أو أكـــثر علـــى حسب وظيفة البرنامج .

ثالثاً الكلمة **void** قبل اسم الدالة تعني أن الدالة ( ) main لن ينتج عنها قيمة تستخدم بعـــد ذلـــك في حسابات أخرى ، أما كلمة void داخل الأقواس فتدل على أن الدالة لا تحتـــاج إلى أي وســـيطات Arguments .



وسيطات الدالة هي المعطيات التي تدخل إليها للحصـول على النتائج.



البرنامج في C يتكون من مجموعة دوال.

الدالة ( )Main هي الدالة التي ينقل إليها التحكم من نظـــام التشغيـــل عند بدء تشغيل البرنامج.

رابعـــاً السطر الثاني في البرنامج هو عبارة عن قوس } وهو يستخدم لأخبار المترجم **Compiler** أنـــه بداية وحدة - **Block** .... وهي مجموعة من الجمل التي يجب أن يستمر تنفيذها حتى نهايتها كما في دالتي **End , Begin** في لغة **Pascal** مثلاً وفي برنامجنا هنالك جملـــة واحـــدة في الوحـــدة -

Block ، هي الجملة التي تبدأ بـــ ( ) printf ووظيفتها تتمثل في طباعة ما بين علامات التنصيص على الشاشة.

خامساً تنتهي الجملة Statement في C بواسطة (;) في نهايتها .

سادساً من الجدير بالذكر أن المترجم Compiler لا يعتد بأي مسافات خالية فيمكنك كتابة البرنـــــامج ككل بالصورة التالية:

```
void main (void) {printf (" I will be a Professional C
Programmer .... Soon \ n");}
```

ولن يكون هناك فارق بالنسبة للمترجم لكن بالطبع الصورة الأولى تكون أفضل لقراءة البرنـــامج بواســـطة المبرمج أو أي شخص يطلع عليه.



المسافات الخالية . Tab . الأسطر الجديدة . ... كل ذلك غير مرئي بالنسبة للمترجم.

نقطة أخرى تستحق الذكر في هذا الموضع وهي أن C تفرق بين الكلمات المكتوبة بأحرف صغيرة Small والكلمات المكتوبة بأحرف كبيرة Capital فمثـــلاً Printf() تختلــف عـــن printf() وتختلــف عـــن PRINTF() وبصفة عامة حافظ على أن تكون كتابتك داخل الكود بأحرف صغيرة ولا تستخدم الأحرف الكبيرة إلا للضرورة القصوى.

سابعاً الدالة printf() وظيفتها ليست موجودة داخل البرنامج الذي تمت كتابته فمــــن أيـــن تفـــهم C مهمتها.... ؟!؟

يأتي ذلك من إحدى المكتبات الموجودة مع C فعندما تقوم بترجمة البرنامج - يضع المترجم Compiler إلى جوار هذه الدالة ما يفيد أنها مرتبطة بمكتبة معينة وبالتالي فإنه عندما تأتي مرحلة الربط Linking لتكويـــن ملف EXE تكون مهمة الرابط Linker هي ربط المكتبة مع ملف المصدر الذي يحتويها والذي أعلنا عنــــه في بداية البرنامج من خلال السطر

```
# include < stdio.h>
```

كل مجموعـة مـن دوال المكتبـات تكـون ضمـن ملـف مصـدر واحد، ولكي يتعـرف عليـها المـترجم نسـتخدم جملـة **#include** لتحديـد اسـم الملـف الـذي يحتـوي علـى المكتبـة فعلـى سـبيل

المثـــال stdio.h يحتـــوي علـــى دوال الإدخـــال والإخـــراج وباســـتخدام include يتـم احتـواء ملـف المصـدر المكتـوب بواسطة المبرمج على ملف المصدر الذي به المكتبات.

ثامناً تعتبر جملة include# هي جملة تعليمات موجهة إلى المترجم Compiler وليس كباقي جمل البرنامج تعليمات موجهة للمعالج فمثلاً سطر مثل

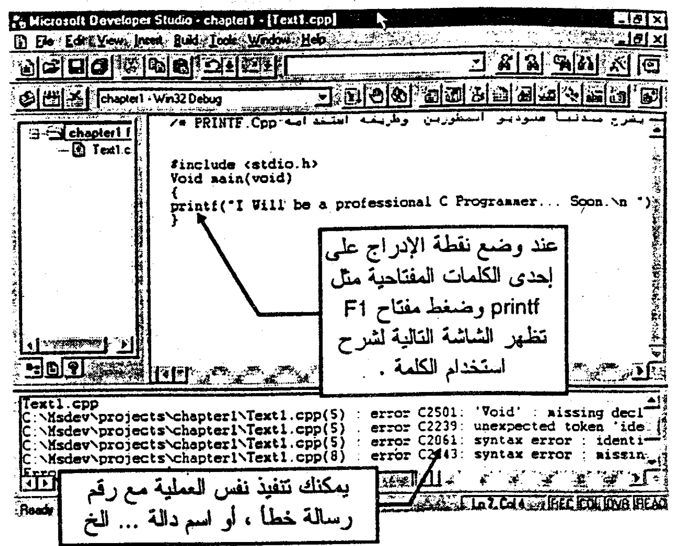
```
i = 5 ;
```

تعني تعليمات للمعالج يقوم بعدها بوضع القيمة 5 في المتغير i لكن السطور التي تبدأ بالرمز # تحتوي على تعليمات موجهة للمترجم نفسه حتى يقوم بإضافة ملف مكتبات أو تغيير من شكل الترجمة حسب الحاجة ، لذلك تسمى الجمل التي تبدأ بهذا الرمز موجهات المترجم وسيلي مناقشتها بالتفصيل في الفصل الرابع.

الحصول على المساعدة

أسرع طريقة يمكنك بها الحصول على المساعدة أثناء العمل هي ضغط مفتاح F1 من لوحة المفاتيح . ببساطة عندما تريد مساعدة على كلمة معينة (جملة - اسم دالة ... الخ) موجودة داخل كود أي برنامج ، ضع نقطة الإدراج فوق هذه الكلمة ثم اضغط F1 ليفتح لك البرنامج مساعدة فورية عليها.

تفيد هذه الطريقة أثناء كتابة أو مراجعة Code أي برنامج أو في حالة ظهور رسائل خطأ في عملية الترجمة Compiling حيث تقف عند رقم رسالة الخطأ وتضغط F1 لفتح مساعدة على طريقة حل هذه المشكلة كما بالشكل التالي:



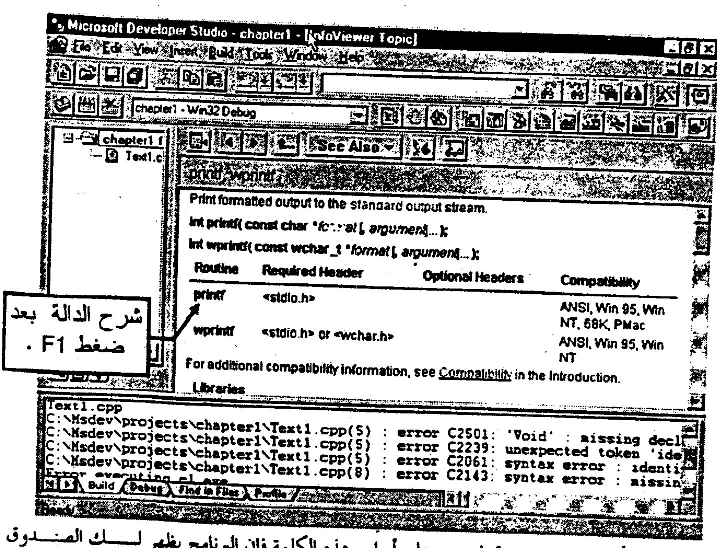


كما يظهر من الشكل السابق فإن المترجم عند وجود خطأ يعرضه في المساحة السفلية من النافذة بحيث يظهر في كل سطر خطأ واحد.

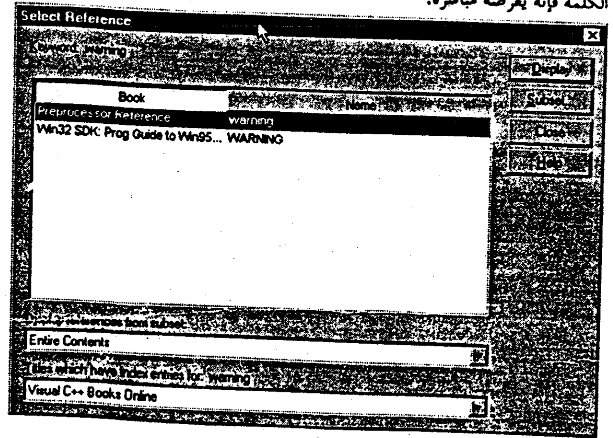
- يظهر في بداية السطر ناحية اليسار موضع الملف واسمه.
- يظهر بعد اسم الملف رقم السطر المحتوى على الخطأ.
- بعدها يظهر رقم الخطأ الذي يمكنك البحث عن المساعدة على أساسه.
- في نهاية السطر تظهر جملة شرح مبسطة عن طبيعة الخطأ.

'Void' missing declared identifier

وهو خطأ ناتج عن وجود حرف **V** كبير **Capital** في كلمة **Void** .

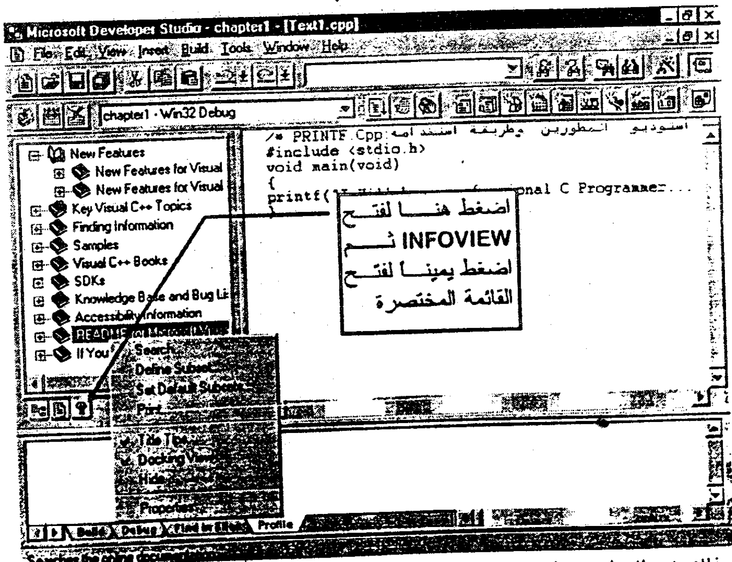


وإذا كان هناك أكثر من موضوع تم فهرسته على أساس هذه الكلمة فإن البرنامج يظهر لـــك الصـــندوق الحواري التالي لتحديد الموضوع بالضبط ثم يبدأ في عرضه أما إذا كان هناك موضوع واحد فقط مفـــهرس على هذه الكلمة فإنه يعرضه مباشرة.



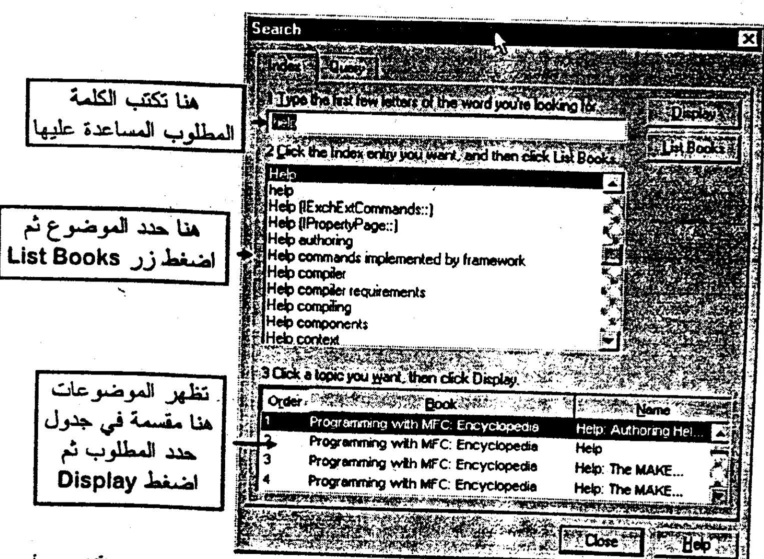
طريقة أخرى سريعة للحصول على مساعدة عن موضوع معين هي الضغط يميناً فوق أي عنصر من عناصر فهرس الموجودة في Info view ، ومن القائمة المختصرة كما بالشكل التالي اضغط Search أو اختـــار الأمر

Help/Search



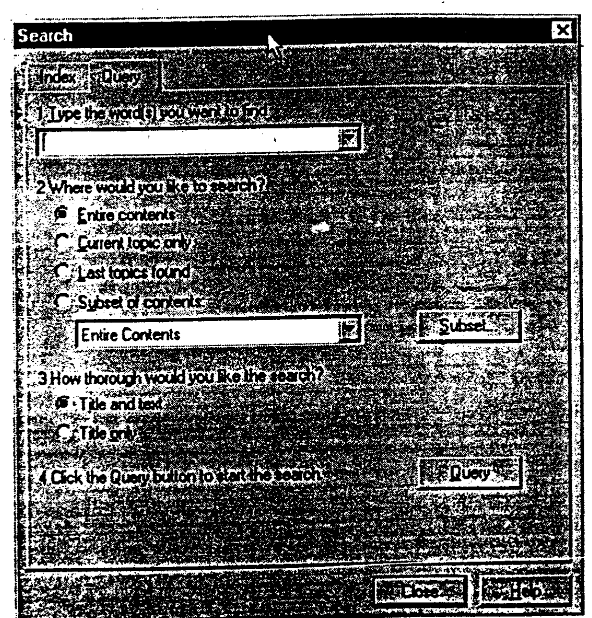
وبذلك يفتح لك البرنامج الصندوق الحواري التالي أكتب الكلمة التي تريد البحث عن معلومات عنـــها ثم اضغط List Books لعرض المواضع التي ستجد فيها معلومات عن هذه النقطة.

ومن القائمة السفلية أضغط الموضوع المطلوب ثم اضغط زر Display لعرض الموضوع في Info View .

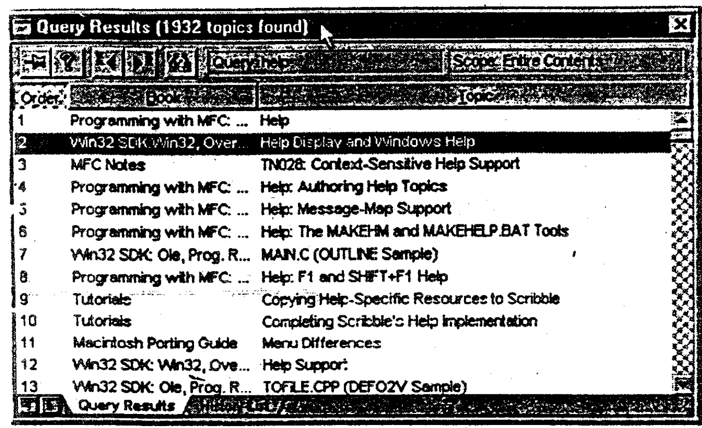


في حالة عدم توافر أي مساعدة عن الكلمة التي اخترتها يوفر البرنامج وسيلة أخرى للبحث عن الكلمة أو الجملة المراد البحث عنها في جميع ملفات المساعدة ، وعرضها في صورة استعلام Query ولكي تقوم بتحضير مثل هذا الاستعلام تابع الخطوات التالية:

1. افتح صندوق Search كما سبق بإحدى الطريقتين سواء أوامر القائمة Help أو بزر أداة التأشير الأيمن.
2. من الصندوق الحواري اضغط Query لفتح الكارت الحواري ليصبح Query على الصورة التالية



3. اكتب الكلمة التي تريد البحث عنها ثم حدد البحث هل يتم في جميع المحتويات أم في الفهرس الذي تقف عنده في info view فقط أو وفق مجموعة فرعية Subset من المحتويات تكون حددتها مسبقاً، حـــدد بعدها هل يتم البحث في العناوين فقط أم في كامل النص.
4. اضغط زر Query للبدء في تجهيز الاستعلام.
5. بعد الانتهاء تستطيع استعراض الموضوعات التي ترتبط بما حددته من خلال النافذة التالية والتي تمثل نتيجة الاستعلام.



وفيها تظهر جميع الموضوعات التي تحتوي على الكلمة أو التعبير الذي ذكرته لعمل الاستعلام على أساسه . حدد الموضوع الذي تريده وبالضغط المزدوج عليه يتم فتحه كما سبق لعرض محتويات الكتب الفورية ن الموضوع المحدد.

References

1. **Visual C++** خوارزم للنشر والتوزيع ، فريق فرسان الإنتاج

2. المرجع الأساسي لمستخدمى سى ، مجدى محمد أبو العطا ، العربية لعلوم الحاسب

3. **Gamma Erich, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.**

4. **Lippman, Stanely B. C++ Primer. Second Edition. Addison-Wesley. 1991.**

5. **Meyers, Scott. Effective C++: 50 Specific Ways to improve your programs and designs. Addison-Wesley, 1992.**

6. **Plauger, P.J. The Draft Standard C++ Library. Prentice Hall PTR, 1995.**

7. **Stroustrup, Bjarne. The C++ Programming Language. Second Edition. Addison-Wesley, 1991.**

8. **Stroustrup, Bjarne. The Design and Evolution C++. Addison-Wesley, 1994.**